

Empowering Large Language Models to Edge Intelligence: A Survey of Edge Efficient LLMs and Techniques

Rui Wang^{a,*}, Zhiyong Gao^a, Liuyang Zhang^a, Shuaibing Yue^a, Ziyi Gao^a

^a*School of Computer and Communication Engineering, University of Science and Technology Beijing (USTB), Beijing, 100083, China*

Abstract

Large language models (LLMs) have showcased exceptional capabilities across various natural language processing (NLP) tasks in recent years, such as machine translation, text summarization, and question answering. Despite their impressive performance, the deployment of these models on edge devices, such as mobile phones, IoT devices, and edge computing nodes, is significantly hindered by their substantial computational and memory requirements. This survey provides a comprehensive overview of the state-of-the-art techniques and strategies for enabling efficient inference of LLMs on edge devices. We explore approaches including the development of small language models (SLMs), model compression techniques, inference optimization strategies, and dedicated frameworks for edge deployment. Our goal is to highlight the advancements and ongoing challenges in this field, offering valuable insights for researchers and practitioners striving to bring the power of LLMs to edge environments.

Keywords:

Large Language Model, Edge Intelligence, Small Language Model, Model Compression, Efficient Inference, On-device LLM

*Corresponding author

Email address: wangrui@ustb.edu.cn (Rui Wang)

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 3 |
| 2 | Small Language Model | 5 |
| 2.1 | Differences and Similarities with Large Models | 7 |
| 2.2 | Off-the-shelf Small Models | 9 |
| 2.2.1 | Small Variants of Large Models | 9 |
| 2.2.2 | Pre-trained Small Models | 12 |
| 2.2.3 | Compressed Small Models | 16 |
| 2.3 | Prospects and Applications | 17 |
| 2.3.1 | SLMs for Specialized Domains | 17 |
| 2.3.2 | Application in Multimodal Models | 18 |
| 2.3.3 | AI Agents for Edge | 18 |
| 2.3.4 | Mobile Assistants | 19 |
| 3 | Model Compression | 19 |
| 3.1 | Pruning | 19 |
| 3.2 | Knowledge Distillation | 20 |
| 3.3 | Quantization | 21 |
| 3.4 | Low-Rank Decomposition | 24 |
| 4 | Inference Optimization | 25 |
| 4.1 | Speculative Decoding | 25 |
| 4.2 | KV Cache Compression | 27 |
| 4.3 | Early Exiting | 28 |
| 4.4 | Kernel Optimization | 30 |
| 4.5 | Memory Offloading | 31 |
| 5 | Deployment | 32 |
| 5.1 | On-device Inference Engines | 32 |
| 5.2 | Cloud-Edge Collaborative Frameworks | 33 |
| 5.3 | Deployment Suites | 35 |
| 6 | Open Challenges and Future Directions | 35 |
| 6.1 | Resource Constraints on Edge Devices | 35 |
| 6.2 | Dynamicity of Edge Environments | 36 |
| 6.3 | Heterogeneity of Edge Devices | 36 |
| 6.4 | Edge-Cloud Collaboration | 37 |

| | | |
|----------|--|-----------|
| 6.5 | Privacy and Security Concerns | 37 |
| 6.6 | Hallucination Issues | 37 |
| 6.7 | Morality and law | 38 |
| 6.8 | Interpretability | 38 |
| 6.9 | User preferences and personalization | 38 |
| 6.10 | Green and sustainable development | 39 |
| 7 | Conclusion | 39 |

1. Introduction

Large language models (LLMs) have shown remarkable effectiveness in a wide range of natural language processing (NLP) tasks, such as machine translation, text summarization, and question answering. These models are typically trained on large corpora of text data, enabling them to produce coherent and contextually relevant text. Over recent years, the capabilities and sizes of these models have grown significantly, as seen in Figure 1, which illustrates the increase in parameter sizes of various models from early versions like GPT-1 [1] and BERT [2] to more recent advancements like GPT-4 [3], BLOOM [4], LLaMA [5] and Llama2 [6].

However, the large parameter sizes of LLMs present significant challenges for their deployment on edge devices. Edge devices, including mobile phones, IoT devices, and edge computing nodes, typically possess limited computational and memory resources. For instance, Llama2-7B [6] inference requires at least 7GB of CPU or GPU memory with INT4 quantization and only achieves 4.5 tokens per second on NVIDIA Jetson AGX Orin [7], which is often beyond the capabilities of most edge devices. Moreover, continuous inference can cause devices to heat up significantly, as some edge or mobile devices are typically passively cooled, which may have a significant impact on performance [8]. This high computational requirement and substantial memory footprint hinder the widespread adoption of LLMs in edge environments, where resources are constrained.

Specifically, the edge deployment of LLMs faces four major challenges due to the resource constraints inherent in edge environments:

1. The rapid growth in the size of LLMs is at odds with the limited memory resources of edge devices.
2. The high computational demands of LLMs clash with the restricted computational resources of edge devices.

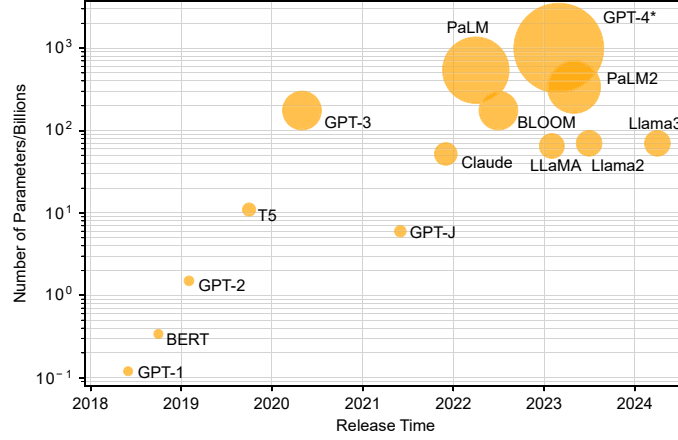


Figure 1: Increase in parameter size of LLMs over time. GPT-4 is estimated to have more than 1 trillion parameters.

3. The substantial energy consumption of LLMs conflicts with the finite energy supply of edge devices.
4. The large throughput requirements of LLMs are in contrast with the limited bandwidth of edge devices.

To address these challenges, existing research has aimed to alleviate resource pressures through: (1) introducing lightweight architectures to reduce model computational complexity and communication overhead; (2) utilizing model compression techniques to decrease the scale of model parameters; (3) optimizing inference system efficiency by designing effective inference strategies and algorithms.

Despite these challenges, deploying LLMs on edge devices presents distinct advantages in latency, privacy, personalization and so on [9]. Edge deployment can significantly reduce latency, as data processing occurs closer to the source, thereby improving real-time responsiveness. It can also enhance data privacy and security, as sensitive information does not need to be transmitted to centralized servers for processing. Additionally, edge deployment can lead to more efficient use of network bandwidth and provide uninterrupted services even in areas with limited connectivity. Furthermore, edge-based LLMs can offer more personalized experiences by leveraging local data to tailor responses and services to individual users, thereby enhancing user satisfaction and engagement. According to Statista’s forecast, the number of globally connected Internet of Things (IoT) devices is expected to reach 15.9 billion by 2023 and is estimated to escalate to

39.6 billion by 2033 [10]. The rapid expansion of edge IoT devices necessitates the exploration of redundant computational power at the edge and the utilization of edge advantages to provide services.

The remainder of this survey is organized as follows: Section 2 provides a comprehensive summary and detailed investigation of recent notable small language models. Section 3 presents the latest advancements in LLM compression techniques. Section 4 delves into research progress on LLM inference optimization technologies for edge computing. Section 5 explores frameworks suitable for deploying LLMs on edge devices. Section 6 highlights the challenges associated with LLMs on the edge and discusses the future. Finally, we conclude the survey in Section 7. Table 1 lists the acronyms used throughout this survey. We hope this survey will serve as a valuable resource for researchers and practitioners working to bring the power of LLMs to edge environments.

2. Small Language Model

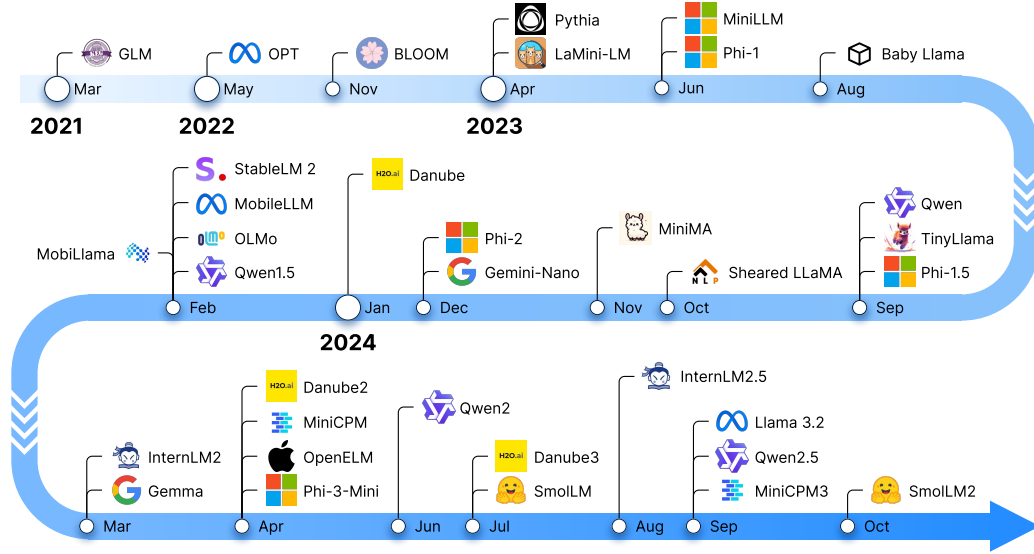


Figure 2: Timeline of small language models from 2022 to 2024.

The edge availability of large language models is essential for a wide range of applications. However, deploying LLMs on edge devices is challenging due to the high computational requirements of these models. Therefore, small language models (SLMs) have been developed in recent years to address this issue, as

Table 1: Abbreviation list.

| Abbreviation | Full Form |
|---------------------|--|
| LLM | Large Language Model |
| SLM | Small Language Model |
| NLP | Natural Language Processing |
| MHA | Multi-Head Attention |
| MQA | Multi-Query Attention |
| MLA | Multi-head Latent Attention |
| IoT | Internet of Things |
| GQA | Grouped-Query Attention |
| FNN | Feedforward Neural Network |
| KLD | Kullback-Leibler Divergence |
| MLLM | Multimodal Large Language Model |
| RAG | Retrieval-Augmented Generation |
| CoT | Chains-of-Thought |
| ICL | Incremental Context Learning |
| IF | Instruction Following |
| PTQ | Post-Training Quantization |
| QAT | Quantization-Aware Training |
| SIMD | Single Instruction Multiple Data |
| KV Cache | Key-Value Cache |
| SRAM | Static Random Access Memory |
| DRAM | Dynamic Random Access Memory |
| GEMM | General Matrix Multiplication |
| SpMM | Sparse Matrix Multiplication |
| FL | Federated Learning |
| HFL | Hierarchical Federated Learning |
| BSBODP | Bridge Sample Based Online Distillation Protocol |
| ASIC | Application-Specific Integrated Circuit |

shown in Figure 2. SLMs are typically smaller in size and have fewer parameters than their larger counterparts, making them suitable for edge devices with limited computational resources. In this survey, we define SLMs as language models with fewer than 4 billion parameters, which is a common threshold for distinguishing between large and small language models, and focus on transformer-based architectures.

We collected information on models submitted to Open LLM Leaderboard [11] and analyzed the distribution of model parameters, as shown in Figure 3. The histogram reveals a significant boundary around 4-6B parameters, indicating that research on small models primarily focuses on models with fewer than 4B parameters. Therefore, in this survey, we define small models as those with fewer than 4B parameters.

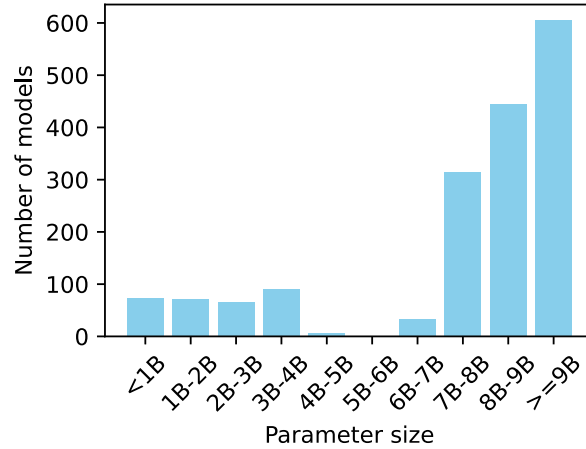


Figure 3: Histogram of model parameters from Open LLM Leaderboard [11].

2.1. Differences and Similarities with Large Models

Modern large language models are typically based on the transformer architecture, which has achieved significant success in the field of natural language processing. The SLMs discussed in this survey typically utilize a transformer architecture similar to LLMs, but differ in parameter size, training dataset, and training methodology. Additionally, to achieve efficient inference, some SLMs adopt special optimization strategies different from those of LLMs. Due to their smaller parameter sizes, SLMs may exhibit reduced capabilities in logical reason-

ing, long-context understanding, and multilingual processing compared to larger LLMs, which excel in these areas.

1. **Parameter Size.** Parameter size is the primary criterion for distinguishing between SLMs and LLMs in this survey. The smaller parameter size of SLMs is mainly due to reduced model layers, hidden dimensions, attention heads, and intermediate dimensions, resulting in faster inference speeds and lower memory consumption compared to larger models.
2. **Training Datasets.** SLMs typically require much smaller training datasets than LLMs, which may result in lower performance on some tasks compared to LLMs. Additionally, many SLMs are trained using synthetic data generated from larger models and high-quality data curated and filtered carefully, as inspired by the research of Phi-1 [12].
3. **Training Algorithms.** While most foundational LLMs are trained from scratch, some SLMs follow this approach as well. However, there are differences in hyperparameter settings, training data preferences at different stages, and other aspects. For example, MiniCPM [13] conducted detailed experiments to select the best hyperparameters. Some SLMs opt to distill or prune existing LLMs to obtain better performance than training from scratch.
4. **Model Architecture.** To achieve efficient inference, many SLMs employ efficient attention mechanisms such as GQA [14] and MQA [15], which reduce time and space complexity compared to MHA [16]. Another common technique in SLMs is parameter sharing, including embedding weight tying [17] and transformer weight sharing [18]. By reducing the model’s parameter size, these techniques enhance both memory and computational efficiency. We also note that some SLMs prefer deeper and thinner architectures in smaller models, rather than reducing the number of layers. This viewpoint was proposed and verified by MobileLLM [18] and has been adopted by many other SLMs.
5. **Application Tasks.** Research on emergent abilities [19] indicates that certain capabilities are only present in LLMs with larger parameter sizes, particularly in tasks requiring complex reasoning or logic, where larger models often have an advantage. However, SLMs can also perform well in specific tasks, such as summarization, instruction following, rewriting, and code generation. Therefore, smaller models are more suitable for certain specific tasks and domain-specific tasks without complex reasoning requirements.

Table 2: An overview of training details for selected small language models.

| Model | Date | Precision | Tokens | Trained Tokens | Hardware | Datasets | Language | Institute |
|-------------|---------|-----------|------------------|-----------------|---------------------------------|------------------------------------|----------------------------|-----------------|
| GLM | 2021-03 | - | - | - | - | Pile | en | THU |
| OPT | 2022-05 | - | 180B | - | - | RoBERTa, Pile, PushShift.io Reddit | en | Meta |
| Bloom | 2022-11 | fp16 | 341B | 410B | 384 A100 80GB | ROOTS | 48 languages | BigScience |
| Phi-1 | 2023-06 | fp16 | 7B | 51B | 8 A100 | CodeTextbook, CodeExercises | en | Microsoft |
| Gemini-Nano | 2023-12 | - | - | - | TPUv4 and TPUv5e | - | multilingual | Google |
| Phi-2 | 2023-12 | fp16 | 250B | 1.4T | 96 A100 80G | - | en | Microsoft |
| TinyLlama | 2023-09 | bf16 | 950B | 3T | 16 A100 40G | SlimPajama, StarCoder | en, zh* | SUTD |
| Phi-1.5 | 2023-09 | fp16 | 30B | 150B | 32 A100 40G | CodeTextbook, synthetic data | en | Microsoft |
| Qwen | 2023-09 | bf16 | - | 2.2T | - | - | en, zh | Alibaba |
| Danube | 2024-01 | fp8 | - | 1T | 8 H100 80G | - | en | H2O.ai |
| MobileLLM | 2024-02 | - | 1T | - | 32 A100 80G | - | en | Meta |
| Qwen1.5 | 2024-02 | bf16 | - | 2.4T | - | - | 12 languages | Alibaba |
| OLMo | 2024-02 | bf16 | 3.0T | at least 2T | 1024 MI250X 128GB, 216 A100 40G | Dolma | en | A12 |
| StableLM 2 | 2024-02 | bf16 | - | 2T | 512 A100 40G | Restruct-v1 | en, de, es, fr, it, nl, pt | Stability AI |
| Gemma | 2024-03 | - | 3T | - | 512 TPUv5e | - | en | Google |
| InternLM2 | 2024-03 | - | 2T | - | - | - | en, zh | Shanghai AI Lab |
| Phi-3-Mini | 2024-04 | bf16 | 3.3T | - | 512 H100 80G | - | en | Microsoft |
| OpenELM | 2024-04 | - | 1.5T | - | 128 A100/H100 80G | RefinedWeb, RedPajama, Pile, Dolma | en | Apple |
| Danube2 | 2024-04 | fp8 | - | 3T | 8 H100 80G | - | en | H2O.ai |
| MiniCPM | 2024-04 | - | 1.1T | - | - | CommonCrawl, Dolma, C4, Pile, etc. | en, zh | OpenBMB |
| Qwen2 | 2024-06 | bf16 | 7T (12T on 0.5B) | - | - | - | 27 languages | Alibaba |
| SmolLM | 2024-07 | bf16 | - | 1T | 64 H100 80G | SmolLM-Corpus | en | HuggingFace |
| Danube3 | 2024-07 | - | - | 6T (4T on 500M) | - | - | en | H2O.ai |
| InternLM2.5 | 2024-08 | - | - | - | - | - | en, zh | Shanghai AI Lab |
| MiniCPM3 | 2024-09 | - | - | - | - | - | en, zh | OpenBMB |
| Qwen2.5 | 2024-09 | bf16 | 18T | - | - | - | 29 languages | Alibaba |
| SmolLM2 | 2024-10 | bf16 | - | 11T | 256 H100 80G | SmolLM-Corpus | en | HuggingFace |

Here, "Date" refers to the release date of the corresponding paper or article. "Precision" denotes the numerical precision used during training, where "bf16" stands for bfloat16 and "fp16" stands for float16. "Tokens" represents the number of tokens in datasets, and "Trained Tokens" denotes the number of tokens used for training. "Hardware" refers to the hardware used for training. "Datasets" lists the datasets used for training. Mentioned datasets are RoBERTa [20], Pile [21], PushShift.io Reddit [22], ROOTS [23], SlimPajama [24], StarCoder [25], CodeTextbook/CodeExercises [12], Restruct-v1 [26], RefinedWeb [27], RedPajama [28], Dolma [29], CommonCrawl [30], C4 [31], WuDao [32], Chook [33], SmolLM-Corpus [34]. "Language" indicates the language(s) of the datasets, some models' language support is too extensive to list here. "Institute" refers to the institution that developed the model.

* These models are monolingual, but have multilingual versions.

2.2. Off-the-shelf Small Models

SLMs can be categorized based on their development approaches into small variants of LLMs, pre-trained SLMs, and compressed SLMs. Small variants of LLMs refer to models that are smaller in size within a series, typically sharing the same architecture, corpus, and training methodology as their larger counterparts. Pre-trained SLMs are designed specifically as smaller LLMs, developed from scratch with architectural and training optimizations to further reduce model resource requirements and enhance performance. Compressed SLMs are derived from larger LLMs through the process of pruning or knowledge distillation, which will be discussed in Section 3. This section will discuss each of these three types of SLMs in detail. We list the training details of some common SLMs in Table 2 and the architecture details in Table 3.

2.2.1. Small Variants of Large Models

OPT [35]: OPT represents a series of open-source pre-trained models that replicate the performance and scale of the GPT-3 [36] model, with parameter sizes ranging from 125M to 175B. The authors provide a detailed discussion on the training process and evaluation results of the OPT models. The hyperparameter settings of the OPT models largely follow the design principles of GPT-3. To prevent underflow, OPT employs dynamic loss scaling during training described in [37].

Table 3: Architecture details for selected small language models.

| Model | Size | Arch. | Atten. | Vocab. | Tokenizer | Norm | PE | Act. | Bias | L | H | d | CL |
|---------------------|---------------|--------|------------|--------|---------------|------------------------------|---------|--------|------|-------|----------------|------|-----------------------------|
| OPT | 125M | - | MHA | 50k | BPE | pre LayerNorm | Learned | ReLU | yes | 12 | 12 | 768 | 2k |
| | 350M | | | | | | | | | 24 | 16 | 1024 | |
| | 1.3B | | | | | | | | | 24 | 32 | 2048 | |
| | 2.7B | | | | | | | | | 32 | 32 | 2560 | |
| Bloom | 560M | - | MHA | 250k | BPE | pre LayerNorm | ALiBi | GeLU | yes | 24 | 16 | 1024 | 2k |
| | 1.1B | | | | | | | | | 24 | 16 | 1536 | |
| | 1.7B | | | | | | | | | 24 | 16 | 2048 | |
| | 3B | | | | | | | | | 30 | 32 | 2560 | |
| GLM | 2B | GPT2 | MHA | 50k | BPE | post LayerNorm | Learned | GeLU | yes | 36 | 32 | 2048 | 1k |
| InternLM2 | 1.5B | - | GQA | 93k | BPE | pre RMSNorm | RoPE | SwiGLU | no | 24 | 16 | 2048 | 32k |
| InternLM2.5 | 1.8B | - | GQA | 93k | BPE | pre RMSNorm | RoPE | SwiGLU | no | 24 | 16 | 2048 | 32k |
| Gemini-Nano | 1.8B 3.25B | - | - | - | SentencePiece | - | - | - | - | - | - | - | - |
| TinyLlama | 1.1B | Llama | GQA | 32k | BPE | pre RMSNorm | RoPE | SwiGLU | no | 22 | 32 | 2048 | 2k |
| Qwen | 1.8B | Llama | MHA | 152k | BPE | pre RMSNorm | RoPE | SwiGLU | qkv | 24 | 16 | 2048 | 8k |
| Qwen1.5 | 0.5B | Llama | MHA | 152k | BPE | pre RMSNorm | RoPE | SwiGLU | qkv | 24 | 16 | 1024 | 32k |
| | 1.8B 4B | | | | | | | | | 24 | 16 | 2048 | |
| Qwen2 | 0.5B | Llama | GQA | 152k | BPE | pre RMSNorm | RoPE | SwiGLU | qkv | 24 | 14 | 896 | 32k |
| | 1.5B | | | | | | | | | 28 | 12 | 1536 | |
| Qwen2.5 | 0.5B | Llama | GQA | 152k | BPE | pre RMSNorm | RoPE | SwiGLU | qkv | 24 | 14 | 896 | 32k |
| | 1.5B 3B | | | | | | | | | 28 | 12 | 1536 | |
| Gemma | 2B | - | MQA | 256k | SentencePiece | pre RMSNorm | RoPE | GeGLU | | 18 | 8 | 2048 | 8k |
| Phi-1 | 1.3B | - | MHA | 51k | BPE | pre LayerNorm | RoPE | GeLU | yes | 24 | 32 | 2048 | 2k |
| Phi-1.5 | 1.3B | - | MHA | 51k | BPE | pre LayerNorm | RoPE | GeLU | yes | 24 | 32 | 2048 | 2k |
| Phi-2 | 2.7B | - | MHA | 51k | BPE | pre LayerNorm | RoPE | GeLU | yes | 32 | 32 | 2560 | 2k |
| Phi-3-Mini | 3.8B | - | MHA | 32k | BPE | pre RMSNorm | RoPE | SwiGLU | no | 32 | 32 | 3072 | 4k, 128k ² |
| StableLM 2 | 1.6B | - | MHA | 100k | BPE | pre LayerNorm | RoPE | SwiGLU | qkv | 24 | 32 | 2048 | 4k |
| OpenELM | 270M-3B | - | GQA | 32k | BPE | pre RMSNorm | RoPE | SwiGLU | no | 16-36 | - ¹ | 1280 | 2k |
| OLMo | 1B | - | MHA | 50k | BPE | pre non-parametric LayerNorm | RoPE | SwiGLU | no | 16 | 16 | 2048 | 2k |
| MiniCPM | 1.2B | - | GQA MHA | 123k | BPE | pre RMSNorm | RoPE | SwiGLU | no | 52 | 8 | 1536 | 4k, 128k ² |
| | 2.4B | | | | | | | | | 40 | 36 | 2304 | |
| MiniCPM3 | 4B | - | MLA | 73k | BPE | pre RMSNorm | RoPE | SwiGLU | no | 62 | 40 | 2560 | 32k |
| MobileLLM | 125M | Llama | GQA | 32k | BPE | pre RMSNorm | RoPE | SwiGLU | no | 30 | 9 | 576 | 2k |
| | 350M | | | | | | | | | 32 | 15 | 960 | |
| SmolLM & SmolLM2 | 135M | Llama | GQA | 49k | BPE | pre RMSNorm | RoPE | SwiGLU | no | 30 | 9 | 576 | 2k (SmolLM) 8k (SmolLM2) |
| | 360M | | | | | | | | | 32 | 15 | 960 | |
| | 1.7B | | | | | | | | | 24 | 32 | 2048 | |
| Danube | 1.8B | Llama2 | GQA | 32k | BPE | pre RMSNorm | RoPE | SwiGLU | no | 24 | 32 | 2560 | 16k |
| Danube2 | 1.8B | Llama2 | GQA | 32k | BPE | pre RMSNorm | RoPE | SwiGLU | no | 24 | 32 | 2560 | 8k |
| Danube3 | 500M | Llama | GQA | 32k | BPE | pre RMSNorm | RoPE | SwiGLU | no | 16 | 16 | 1536 | 8k |
| | 4B | | | | | | | | | 24 | 32 | 3840 | |
| Sheared LLaMA | 1.23B | Llama | GQA | 128k | BPE | pre RMSNorm | RoPE | SwiGLU | no | 16 | 32 | 2048 | 128k |
| | 3.21B | | | | | | | | | 28 | 24 | 3072 | |
| Llama 3.2 | 1.3B | Llama | MHA | 32k | BPE | pre RMSNorm | RoPE | SwiGLU | no | 24 | 16 | 2048 | 4k |
| | 2.7B | | | | | | | | | 32 | 20 | 2560 | |

Here, "Size" refers to the number of parameters in the model. "Arch." denotes the basic architecture used in the model. "Atten." represents the attention mechanism used in the model, where "MHA" stands for multi-head attention, "GQA" stands for generalized query attention and "MQA" stands for multi-query attention. "Vocab." denotes the size of the vocabulary. "Tokenizer" refers to the tokenizer used for the model. "Norm" denotes the normalization layer used in the model. "PE" represents the positional embedding used in the model. "Act." denotes the activation function used in the model. "Bias" indicates whether the model uses bias in the attention mechanism. "qkv" means only the QKV layer of attention have bias. "L" represents the number of layers in the model. "H" denotes the number of attention heads. "d" represents the hidden dimension of the model. "CL" denotes the context length of the model.

¹ OpenELM has different numbers of attention heads for each layer.

² Models with long context support are another independent model.

BLOOM [4]: BLOOM is a multilingual open-source LLM that was trained on the ROOTS corpus [23]. To mitigate training instabilities, an additional layer normalization step was introduced after the first embedding layer. BLOOM is avail-

able in six versions, with parameter sizes ranging from 560M to 176B. The authors provide a comprehensive discussion on the creation of the ROOTS training dataset, as well as the architecture and design of the tokenizer used in BLOOM.

GLM [38]: GLM introduces a General Language Model (GLM) based on autoregressive blank infilling. This architecture modifies the masking mechanism of the causal decoder to enable bidirectional attention for prefix tokens while maintaining unidirectional attention for generated tokens. The released pre-trained model is a bilingual LLM. Additionally, another project, ChatGLM-6B [39], is a small-scale popular bilingual dialogue model based on the GLM architecture. By integrating model quantization techniques, it allows for local deployment on consumer-grade graphics cards.

InternLM Series [40]:

InternLM2 [40] is a highly capable open-source LLM, available in various sizes including 1.8B, 7B, and 20B, trained on a diverse dataset and optimized for long-context modeling and robustness. It demonstrates significant advantages in modeling long contexts and conducting open-ended subjective evaluations. The pre-training dataset of InternLM2 encompasses a diverse range of data types, including text, code, and long-context data. To better support long contexts, InternLM2 employs GQA [14] to reduce inference costs and has been additionally trained to accept up to 32k tokens of context.

InternLM2.5 [41] is the next-generation iteration of InternLM2, trained on a large amount of synthetic data. It supports a better long-context and tool usage capability, and achieves advanced levels of mathematical reasoning.

Qwen Series [42, 43, 44]:

The architecture of the Qwen [42] model is similar to that of the LLaMA [5] series. Qwen family encompasses a diverse array of models, including general-purpose, multimodal, and fine-tuned models, extensively trained on up to 3 trillion tokens from a wide range of texts and code, covering a broad spectrum of domains. Notably, the Qwen series excels as a multilingual model, particularly demonstrating outstanding proficiency in both Chinese and English. The Qwen-1.8B model, a smaller variant within the series, delivers competitive performance across various tasks, occasionally surpassing larger models in specific scenarios.

Qwen1.5 [43] significantly enhances the model’s capabilities in multilingual processing, alignment with human preferences, and understanding of long contexts. The smaller variants of Qwen1.5 include models with 0.5B, 1.8B, and 4B parameters, along with several different quantized versions. Compared to leading small models in the industry, Qwen1.5 demonstrates strong competitive performance. Starting from this generation, the smaller Qwen models utilize embedding

tying techniques to reduce the parameter size.

Qwen2 [44] introduces two smaller-sized versions: 0.5B and 1.5B. The training dataset of Qwen has been expanded to include 27 additional languages beyond English and Chinese. Qwen2 features enhanced support for longer context lengths and exhibits significant improvements in coding and mathematical capabilities. Across a spectrum of competencies including natural language understanding, knowledge, coding, mathematics, and multilingual processing, Qwen2 significantly surpasses its predecessors.

Qwen2.5 [45] demonstrates significant improvements in instruction following, long-text generation, long-context understanding, structured data understanding, and structured output generation.

2.2.2. *Pre-trained Small Models*

Phi Series [12, 46, 47, 48]:

Phi-1 [12] asserts that high-quality training data significantly impacts performance metrics. The model utilizes a blend of network data and synthetic data for training, with the synthetic data originating from GPT-3.5 responses. Phi-1 operates on a dataset of less than 7B tokens, surpassing nearly all open-source models on coding benchmarks at the time. However, Phi-1, having been trained solely on Python code, serves as a specialized SLM for coding, which limits its versatility. The original text also points out that Phi-1 lacks the domain-specific knowledge of larger models and exhibits poor robustness to style changes or errors in the prompts.

Phi-1.5 [46] builds upon the insights gained from Phi-1, with a particular focus on the more challenging task of common sense reasoning. Trained on a "textbook-quality" synthetic dataset of 300 billion tokens, Phi-1.5 demonstrates common sense reasoning capabilities comparable to models an order of magnitude larger. This underscores the importance of data quality, which appears to be more critical than previously thought. Additionally, the research suggests that the use of "textbook-like" synthetic data for training seems to mitigate the generation of toxic content compared to models trained solely on internet data.

Phi-2 [47] is a model with 2.7 billion parameters that exhibits outstanding capabilities in reasoning and language understanding. Adhering to the principle that data quality is paramount, the training data for Phi-2 is a blend of a synthetic dataset specifically created for teaching the model common sense reasoning, and carefully selected network data to further enhance the training corpus. Evaluation results reveal that Phi-2's performance in areas such as common sense reasoning, language understanding, mathematics, and coding is comparable to models five

times its size.

Phi-3-Mini [48] employs a block structure similar to that of Llama2 [6] in contrast to Phi-1 and Phi-2. This model, with 2.7 billion parameters, is trained on a colossal dataset of 3.3 trillion tokens. The training data for Phi-3-Mini continues to comprise rigorously filtered public network data and synthetic data generated by LLM. Despite its smaller size, Phi-3-Mini exhibits capabilities in common sense and logical reasoning comparable to Mixtral 8x7B [49] and GPT-3.5, and it is more readily deployable on edge devices. However, the report notes that due to the model’s size constraints, Phi-3-Mini lacks the capacity to store extensive "factual knowledge". Consequently, it still faces challenges in areas such as hallucination, bias, and the generation of inappropriate content.

Gemma [50]: Gemma, a lightweight model released by Google’s DeepMind team, comes in two sizes: 2B and 7B parameters. These models demonstrate robust text understanding and reasoning capabilities. The 2B version is trained on a dataset of 3 trillion tokens, which primarily consists of English data from web documents, mathematics, and code. Evaluations reveal that Gemma not only excels in dialogue and reasoning tasks but also outperforms competitors on six standard safety benchmarks and in human side-by-side evaluations.

TinyLlama [51]: TinyLlama is a compact small language model with 1.1 billion parameters. It employs the same architecture and tokenizer as Llama2 [6] and is pre-trained on a mixed dataset of 30 trillion tokens of natural language and code data. Despite its relatively small size, TinyLlama surpasses OPT-1.3B [35] and Pythia-1.4B [52] in various downstream tasks. Its compactness makes it suitable for end-user applications on mobile devices and for various applications or research studies that require limited computation and memory usage.

MobileLLM [18]: In the pursuit of energy and memory efficiency, MobileLLM investigates use cases for LLMs with fewer than one billion parameters on-device. The study emphasizes the considerable importance of model architecture for LLMs of this scale. It demonstrates that for smaller LLMs, depth is more crucial than width, leading MobileLLM to adopt a deeper and thinner architecture rather than a shallower and wider one. MobileLLM employs GQA [14] and a cache-efficient weight sharing strategy where duplicating the decoding block twice can enhance accuracy. Furthermore, it reuses the input embedding weights as the output fully connected layer weights, resulting in a more efficient and compact model architecture.

MobiLlama [53]: MobiLlama presents accurate yet efficient small language models of 0.5B and 0.8B parameters, optimized for resource-constrained devices. In a significant architectural decision, MobiLlama employs a shared Feed Forward

Network (FFN) design for all blocks in the model, resulting in a 60% reduction in the total number of trainable parameters compared to non-shared models. Despite these reductions, MobiLlama achieves performance comparable to OLMo-1.17B [54] and TinyLlama-1.1B [51]. Remarkably, it does so with significantly less training data and training time, and substantially reduced resource requirements.

MiniCPM Series [13]:

MiniCPM [13] introduces two model sizes, 1.2B and 2.4B, and employs a shared input and output embedding layer, GQA [14], and a deep-and-thin network structure. The research explores the optimal training configuration for SLMs, proposing a three-stage learning rate scheduler, which consists of warmup, stable training and decay stages. During the decay stage, higher quality supervised fine-tuning data is used to promote more comprehensive model learning. Finally, evaluations reveal that MiniCPM possesses capabilities comparable to 7B-13B LLMs, demonstrating its efficiency and effectiveness.

MiniCPM3 4B [55] offers more advanced capabilities than its predecessors, supporting function call, code interpreter, instruction following ability, long context ability, and enhanced reasoning ability. The model has 4B parameters, employs MLA [56] to simultaneously acquire the advantages of GQA [14] and MHA [16], and performs comparably to many models larger than 7B across most benchmarks.

OpenELM [57]: OpenELM, a compact language model with 1.1 billion parameters, was trained and evaluated on public datasets. The core of OpenELM is a layer-wise scaling strategy, which employs smaller hidden dimensions in the attention and feed-forward networks of the earlier layers, gradually increasing the hidden dimensions in the later layers. As a result, each layer in the model has a different number of parameters. This strategy allows OpenELM to make better use of the available parameter budget, achieving higher accuracy. With a parameter budget of approximately one billion, OpenELM’s accuracy improves by 2.36% compared to OLMo [54], while requiring half the pre-training tokens.

OLMo [54]: OLMo introduces a comprehensive framework, encompassing everything from the model, data, training, to evaluation tools, along with detailed metrics collected during training runs. The model comes in two sizes: 1B and 7B. Architecturally, OLMo employs a non-parametric layer norm to achieve faster speed than RMSNorm [58]. It utilizes their open-source pre-training dataset, Dolma [29], and data processing and analysis tools, training on at least 2 trillion tokens using a linear learning rate decay schedule. Remarkably, both the 1B and 7B models surpass their same-scale competitors in zero-shot scores across multiple tasks.

Stable LM 2 [26]: The study focuses on training a small-scale large language model in a reproducible manner, providing a complete and transparent set of principles for designing pre-training datasets and their sources. Experimental results reveal that Stable LM 2 significantly outperforms other similarly-sized open-source models across various tasks and languages. Additional measurements of the model’s throughput performance on edge devices show that it achieves approximately 50+ tokens per second on their consumer-grade products. Notably, using lower precision can double the throughput.

Pythia [52]: Pythia, a suite of models primarily designed for scientific research, encompasses 12 models of varying sizes to better understand model behavior in terms of training and scaling. All models are pre-trained on the same dataset in the same order. Despite the use of several techniques that could potentially impair the performance of smaller models, Pythia’s smaller models still achieve performance comparable to OPT [35]. Additionally, the study provides three case analyses summarizing the impact of data bias on learned behaviors, the influence of training order on memorization, and the effect of pretraining term frequencies on task performance.

SmolLM Series [34, 59]:

SmolLM [34] is an excellent small model released by the HuggingFace. The small-sized model adopts a similar architecture to MobileLLM [18]. They curated and constructed a high-quality dataset called SmolLM-Corpus, which consists of training data from Mixtral [49] synthetic data, educational web content, and filtered Python code. Evaluation results of the model demonstrate advanced performance in common sense reasoning and world knowledge, with the 1.7B model showing strong Python coding performance. SmolLM proves through experiments that small models can achieve good performance as long as they are trained sufficiently and the data quality is good.

SmolLM2 [59] builds upon its predecessor by adopting a more diverse dataset, resulting in significant improvements in instruction following, knowledge, reasoning, and mathematics.

Danube Series [60, 61]

H2O-Danube-1.8B [60] is the foundational model trained by H2O.ai, adopting an architecture similar to Llama2 [6] and delivering performance comparable to Qwen and Stable LM 2 [26]. H2O-Danube2-1.8B [60] inherits the architecture and weights from H2O-Danube-1.8B, continuing training on 2T tokens to achieve higher accuracy, better performance, and long-context support. H2O-Danube3-4B [61] is trained from scratch on a curated 6T token dataset. In addition to its compact size and improved performance, this model outperforms competitors

across various academic benchmarks.

2.2.3. *Compressed Small Models*

Contrary to training a SLM from scratch, model compression methods transfer the capabilities of pre-trained large models to more compact and efficient small architectures for deployment in resource-constrained environments. The most common techniques include pruning and distillation, both of which can preserve specific performance of the original model to a certain extent, and even achieve better compression results by combining multiple methods. There have been many practices generating small models using such methods, and this section will introduce these models.

Gemini-Nano [62]: Gemini-Nano, a member of Google’s Gemini Family, is designed as a series of small models for on-device applications. Available in two sizes, 1.8B and 3.25B parameters, these models are specifically aimed at devices with low and high memory capacities, respectively. The training process involves distillation from larger models within the Gemini series. Impressively, Gemini-Nano models exhibit remarkable capabilities across a variety of tasks, including reasoning, STEM, coding, multimodal, and multilingual tasks.

LaMini-LM [63]: LaMini-LM employs Instruction Following distillation to offer a spectrum of SLMs varying in size and architecture. To accomplish this, a large-scale offline-distillation instruction dataset comprising 2.58 million examples was created, utilizing GPT-3.5-Turbo to generate responses for each instruction. In a comprehensive evaluation, including automatic assessments of downstream NLP tasks as well as human evaluations of general usage, hallucinations, and toxicity, these models achieved superior performance compared to previous methods.

Baby Llama [64]: Baby Llama, a contestant in the BabyLM challenge [65], employs Instruction Following distillation to train a compact teacher model on a dataset comprising 10 million words. This process culminates in the distillation of an even smaller LLaMA model with 58 million parameters. Remarkably, on a majority of zero-shot tasks, this distilled model outperforms its teacher model. The findings of this study suggest that distillation, when the teacher model is trained on a sufficiently small dataset, not only preserves the full performance of the teacher model but can also lead to enhanced performance.

MiniLLM [66]: MiniLLM introduces a white-box knowledge distillation method for LLMs that minimizes the reverse Kullback-Leibler Divergence (KLD) to prevent the student model from overestimating low-probability regions in the teacher model’s distribution, thereby enhancing the quality of generated samples. Exper-

iments conducted on multiple small-sized variants from three different architectures serving as student models demonstrate that MiniLLM achieves lower exposure bias, better calibration, and higher performance in long text generation, along with commendable generative diversity. The student models range in size from 120M to 7B parameters.

MiniMA [67]: reveals a linear correlation between the optimal teacher model size and the student model size across different model architectures and data scales, with the optimal student model size being 0.4 times that of the teacher model. Leveraging this law, a highly computationally efficient 3B model was developed through distillation from the LLaMA2 7B [6] model. Experiments demonstrate that MiniMA sets a new computational performance Pareto frontier among existing 3B models on common benchmarks. Its instruction-finetuned version outperforms 3B competitors in the GPT4 evaluation and is even comparable to several 7B chat models.

Sheared LLaMA [68]: ShearedLLaMA proposes an economically efficient structured pruning method to develop small yet competitive language models. This method first prunes from the LLaMA2-7B model and then trains using only 50 billion tokens. It also introduces dynamic batch loading technology to dynamically adjust the proportion of data from different domains, achieving good accuracy recovery across different domains in the recovery phase after pruning.

Llama 3.2 [69]: Llama 3.2 introduces lightweight text models suitable for edge and mobile devices, with sizes of 1B and 3B. The Llama Team employs pruning and distillation methods to generate these small models. Initially, structured pruning is used to create 1B and 3B initial models from the Llama 3.1 8B model. Subsequently, knowledge distillation is applied, with the Llama 3.1 8B and 70B models serving as teacher models, to restore the performance of the pruned models. In the post-training stage, the team also conducts synthetic data and long-context training on the models. Llama 3.2 also provides quantized versions of these models, with quantization schemes optimized for Arm CPUs, making the models inherently suitable for Arm-based edge hardware devices.

2.3. Prospects and Applications

2.3.1. SLMs for Specialized Domains

Even though SLMs do not possess the powerful generalization capabilities of LLMs, they often exhibit comparable performance in certain downstream tasks or specialized domains. Given specific task requirements, SLMs offer advantages such as low resource consumption, high throughput, low latency, low power consumption, privacy protection, and personalization. Therefore, they hold promising

application prospects in scenarios involving mobile devices, IoT devices, and edge computing devices.

Similar to the application of LLMs in specialized domains, these domain specialized LLMs are typically fine-tuned from base models. The most common application scenarios in specialized domains are code generation and mathematical reasoning. We can already see model variants fine-tuned for these two domains in many pre-trained models, such as DeepSeek-Coder [70] CodeGemma [71], Orca-Math [72], etc. In addition, some models are specifically trained for these domains as well [73, 46]. In other domains, some general small models also demonstrate their advantages in specific areas, such as law and economics [74]. However, whether smaller models have competitive application prospects in these domains remains to be further researched, as most domain specialized LLMs, even at their smallest sizes, reach 7/8B parameters.

2.3.2. Application in Multimodal Models

Due to the efficiency of small text base models, they often play a core component role in some multimodal large language models (MLLM) to achieve efficient inference on mobile SoCs, such as MobileVLM [75], Mobilevlm v2 [76], TinyLLaVA [77], LLaVA-Phi [73], Bunny [78], DeepSeek-VL [79], Vary-toy [80], MiniCPM-V [81], etc. For these types of models, researchers only need to train the mapping relationship between the text representation space and different modality representation spaces (such as vision, audio, video, etc.). Since the SLM contributes the majority of the parameters to the MLLM, its selection is closely related to the lightweight characteristics of this MLLMs [82].

2.3.3. AI Agents for Edge

AI agents refer to systems that can interact with environment on behalf of a user, automate the utilizing of tools, and information collecting . The ability to make function calls is essential for implementing AI agents. In this direction, Octopus has gone further. The Octopus series is a specialized AI agent model tailored for edge devices. Unlike chat models, these models focus more on providing API call capabilities, which are crucial for integrating LLMs externally. To this end, Octopus [83] has designed a dataset from software API documents and condition masking techniques to enhance the effectiveness of LLMs in API interactions. Octopus v2 [84] is specifically trained for Android APIs, facilitating edge computing devices by introducing functional tokens rather than RAG-based methods. Octopus v3 [85] equips the series with multimodal capabilities and compact parameters of less than 1B, enabling good performance on resource-constrained

devices such as Raspberry Pi. Octopus v4 [86], the latest version in the series, offers query redirection functionality. With this, the Octopus team has devised a concept of models that redirect different queries to different vertical domains, achieving optimal performance through multi-node reasoning.

2.3.4. *Mobile Assistants*

More and more mobile device manufacturers have been embedding small language models into mobile smartphones and other devices. Recent practices include Apple’s Apple Intelligent [87] and Google’s Gemini [62]. In addition to industrial practices, many research works are also exploring how to deeply integrate these language models into mobile devices to provide more convenience. AppAgent [88] has developed a multimodal intelligent framework based on LLMs, enabling agents to mimic human interaction styles and operate smartphone applications. MM-Navigator [89] has designed an intelligent system based on GPT-4V, focusing on smartphone graphical user interface navigation tasks, featuring advanced screen understanding and precise action localization capabilities. AutoDroid [90] has crafted a task automation system for Android mobile devices driven by large language models, which can handle any task on any Android application by integrating LLMs with application-specific knowledge, eliminating the need for manual operation. GptVoiceTasker [91] is a virtual assistant capable of comprehending user commands and executing corresponding device interactions, enhancing the user experience and task efficiency on mobile devices, and continuously learning from historical user commands to further improve execution efficiency.

3. Model Compression

Model compression reduces model size by removing redundant information. This can be achieved through pruning, quantization, distillation, or low-rank decomposition. In this section, we provide an overview of these techniques.

3.1. *Pruning*

Pruning is a technique for model compression that eliminates redundant structures or weights, thereby compressing the model. Depending on the granularity, existing techniques for pruning LLMs can be categorized into structured and unstructured pruning. We illustrate five granularities of pruning in Figure 4. Structured pruning [92, 93, 68, 94, 95, 96, 97] involves removing structured components, such as multiple channels or blocks from the parameter matrices of LLMs.

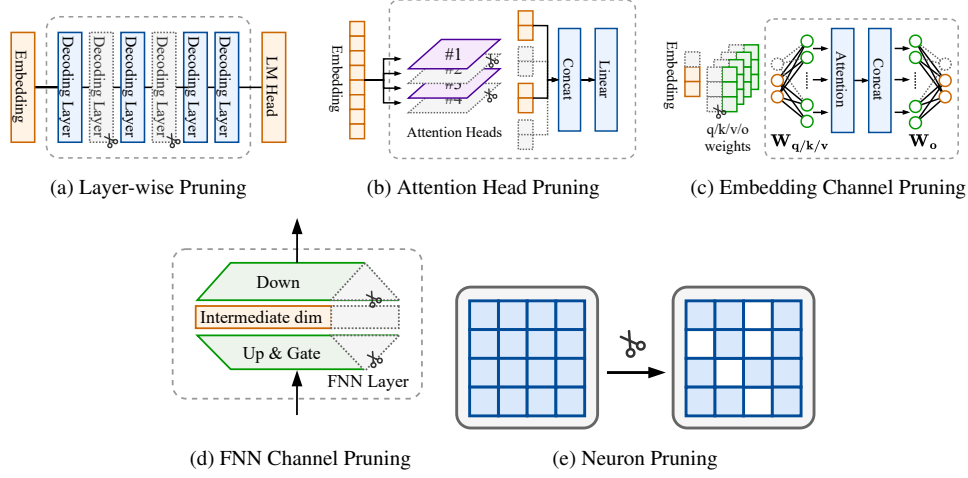


Figure 4: Five granularities of pruning for large language models. Figures 4a, 4b, 4c, and 4d indicate structured pruning, while Figure ?? indicates unstructured pruning.

Some studies [93, 68, 96, 95] have combined fine-tuning, quantization, or even training to mitigate the loss of accuracy and enhance compression efficiency. Conversely, unstructured pruning [98, 99, 100] does not consider the internal structure of LLMs. It induces sparsity in the weight matrix by pruning at the neuron level, which necessitates specialized hardware for accelerating tensor operations. After pruning, the performance of the model is typically restored through fine-tuning. However, due to the huge computational cost associated with full-parameter fine-tuning of LLMs, current large model pruning techniques often forego the fine-tuning step [101, 98] or incorporate parameter-efficient fine-tuning to mitigate these costs [93, 95]. Given that pruning inevitably leads to a loss in model performance and full-parameter fine-tuning of the model is challenging, the practicality of such methods for LLMs remains to be enhanced.

3.2. Knowledge Distillation

Knowledge distillation techniques treat LLMs as teacher models, using the supervisory information of the teacher model to train a smaller student model. Existing research on LLMs can be classified into white-box and black-box distillation, as shown in Figure 5. White-box distillation methods [66, 102, 67, 103] use both the internal information and output of the large model to train the student model, whereas black-box distillation methods [104, 105, 106, 107, 108, 109, 110] assume that the internal structure of the large model is invisible and only use the

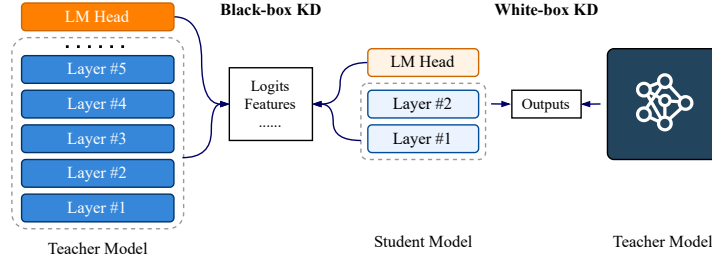


Figure 5: Illustration of knowledge distillation techniques.

output of the teacher model to train the student model. Unlike the knowledge distillation of general neural networks, LLM distillation focuses more on the transfer of knowledge rather than compression of the architecture [111]. When the parameter size of the large model reaches a certain level, it exhibits "emergent abilities", i.e., astonishing performance in handling complex tasks. This feature can help small models learn to cope with complex tasks, thereby giving rise to black-box distillation methods based on Chains-of-Thought (CoT) [104, 105, 106, 107, 108], Incremental Context Learning (ICL) [112], and Instruction Following (IF) [109, 110, 63] capabilities. The knowledge distillation of LLMs is usually used to distill domain knowledge into small models that edge devices can bear, for specific downstream tasks [102]. However, the expressive power of small models is limited compared to LLMs, and users need to further balance between model capability and model size.

3.3. Quantization

Model quantization methods convert the floating-point representation of weights or activation values into lower-precision numerical representations, making full use of the numerical representation space while minimizing errors. The mainstream quantization schemes are Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT). PTQ directly converts the weights of the trained model into a low-precision format without modifying the model architecture or retraining, offering simplicity and efficiency compared to QAT [113]. QAT integrates the quantization process into the model training process, enabling the model to adapt to the low-precision storage format and achieve lower precision loss. Several representative quantization methods are shown in Table 4. Recent works like LLM-QAT [114], EfficientQAT [115] have made efforts to minimize the burden of retraining through knowledge distillation and partial retraining. Another work,

EdgeQAT [116], has designed a QAT method for lightweight LLMs on edge devices which guided by entropy and distribution.

Despite the efforts of the aforementioned studies, the cost of retraining LLMs remains substantial. Consequently, PTQ has emerged as the mainstream technique for quantizing LLMs [117]. The application of PTQ quantization technology in LLMs includes two mainstream directions: weight-only quantization and weight activation value quantization.

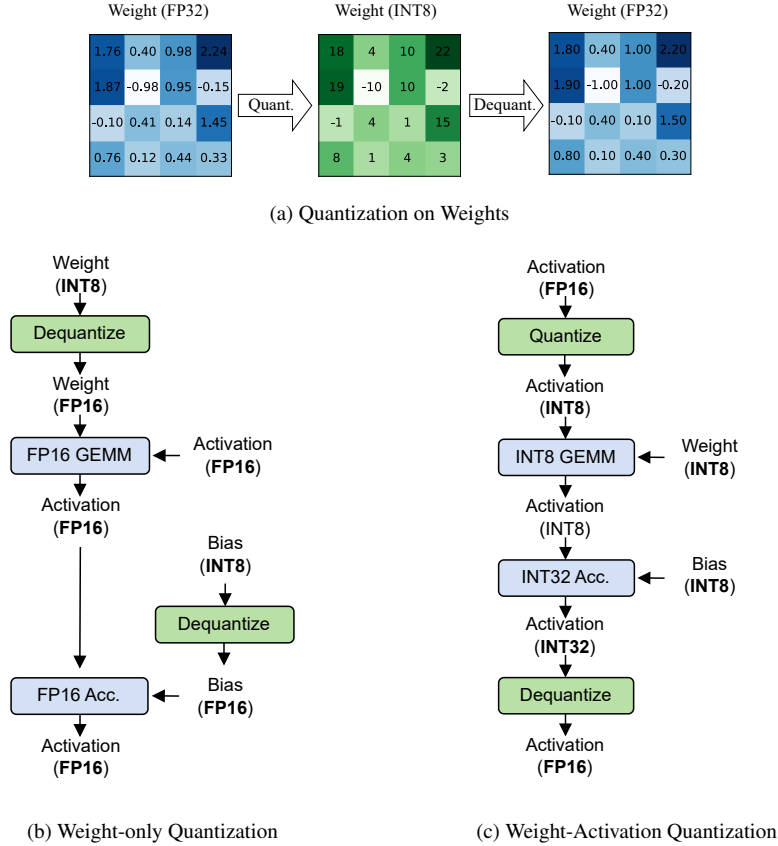


Figure 6: An illustration of quantization before and after, as well as the inference schema for weight-only quantization and weight-activation quantization.

Weight-only Quantization. Weight-only Quantization quantizes only the weights of the model. The low-precision weights are then dequantized to full precision to compute with activation values. Although weight-only quantization does not accelerate LLM inference computationally, it reduces the storage

size of weights, thereby reducing memory bandwidth requirements during matrix loading and improving LLM inference speed. Many works [118, 119, 117, 120, 121] are related to weight-only quantization, some of them were inspired by several insights into compensating for the quantization errors, like outlier separation [122], second-order approximation compensation [118], and distribution smoothing [123] etc. These methods are not always mutually exclusive, and can be combined. For example, SpQR [119] proposed further optimization strategies for the quantization scheme of GPTQ [118], separating outliers and using sparse matrix storage, and adopting a mixed-precision double-layer quantization strategy for non-outlier weights, further reducing the performance loss of LLMs after quantization. AWQ [117], based on the viewpoint of the imbalance of weight importance in LLM, selects important weights according to activation values and introduces a smoothing factor to reduce the quantization error of important weights, ultimately achieving excellent quantization schemes suitable for various LLMs. OWQ [120] theoretically analyzed the amplifying effect of outlier activation values on weight quantization error, and introduced a mixed-precision quantization scheme for the weight matrix based on AWQ. In addition to the aforementioned methods, AQLM [121] combines the advantages of vector quantization and multi-codebook quantization. It represents vectors as the sum of multiple codebook vectors, a process known as Additive Quantization. This allows for a more precise approximation of the original data. AQLM is an asymmetric, non-linear vector quantization method, and its codebooks require a calibration process to learn the optimal solution.

Weight-Activation Quantization. Weight-activation quantization quantizes both weight matrices and activation values. This method reduces memory bandwidth requirements and improves the inference speed of LLMs by replacing high-precision operators with low-precision operators. For some representative works, LLM.int8() [122] discovered the emergence of outlier features and proposed mixed-precision method to preserve high-precision outliers. ZeroQuant [124] proposed a fine-grained hardware-friendly quantization scheme, which uses different quantization granularities for weights and activation values, and employs layer-wise knowledge distillation to mitigate the accuracy loss after quantization. SmoothQuant [123] smooths the distribution of activation values, shifting the difficulty of activation value quantization to model weight quantization, and on this basis, implements a W8A8 quantization scheme for LLMs. Outlier Suppression+ [125], based on Outlier Suppression [126], combines the characteristics of asymmetric outlier distribution and concentration in specific channels, and uses channel-level transformation and scaling to alleviate the errors caused by asymmetric outliers. OliVe [127]

adopts outlier-victim pair quantization, considering that outliers are more important than normal values, and handles local outliers with methods that minimize hardware overhead. QLLM [128] proposed an adaptive channel reorganization method to effectively handle outliers in activation values, and uses calibration data to offset quantization errors. FPTQ [129] designed a novel W4A8 post-training quantization method, combining the advantages of W8A8 and W4A16, and integrating fine-grained weight quantization with layer-wise activation quantization strategies, further maintaining the original performance of the model. Agile-Quant [130] introduces a new quantization framework for LLMs on edge devices, incorporating an activation-aware token pruning technique to mitigate outliers and adverse effects on attention. Agile-Quant achieves end-to-end acceleration across multiple edge devices by leveraging a SIMD-based 4-bit multiplier and an efficient TRIP matrix multiplier.

Binary Quantization. In addition to the aforementioned low-precision quantization schemes, some studies focus on improving the performance of binary neural networks in LLMs. For instance, PB-LLM [131] employs partial binarization to avoid performance degradation caused by sailent weight binarization, and restores the performance by employing PTQ and QAT. BiLLM [132], on the other hand, preserves the residual binary matrix of weights to reduce the quantization error of sailent weights. BitNet [133] introduces a new binary Linear layer and trains the binary LLM by QAT method. Building on this, BitNet b1.58 [134] introduces a new negative value, exploring the performance of LLMs in a ternary system.

3.4. Low-Rank Decomposition

Low-rank decomposition leverages the low-rank characteristics of the model weight matrix, approximating the matrix as two or more smaller matrices to save on the number of parameters. This technique has been widely used for efficient parameter fine-tuning of LLMs [135], but recent work has shown that it can also be used for model compression [136, 137, 138], with excellent compression results. For example, TensorGPT [138] uses low-rank tensor compression for the embedding layer, reducing the spatial complexity of LLMs and making them usable on edge devices. LoSparse [137] approximates the weight matrix as the sum of a low-rank matrix and a sparse matrix, combining the advantages of low-rank approximation and structured pruning, achieving substantial memory savings.

Table 4: Comparison of several representative LLM quantization methods.

| Method | Weight | Activation | Uniform | Symmetric | Calibration | Mixed Precision |
|---------------------|--------|------------|---------|-----------|-------------|-----------------|
| LLM-QAT | ✓ | ✓ | ✓ | ✓ | - | × |
| EfficientQAT | ✓ | × | ✓ | × | - | × |
| EdgeQAT | ✓ | ✓ | ✓ | ✓ | - | ✓ |
| GPTQ | ✓ | × | ✓ | × | ✓ | × |
| SpQR | ✓ | × | ✓ | × | ✓ | ✓ |
| AWQ | ✓ | × | ✓ | × | ✓ | × |
| OWQ | ✓ | × | ✓ | × | ✓ | ✓ |
| AQLM | ✓ | × | × | × | ✓ | × |
| LLM.int8() | ✓ | ✓ | ✓ | × | × | ✓ |
| ZeroQuant | ✓ | ✓ | ✓ | ✓ | × | × |
| SmoothQuant | ✓ | ✓ | ✓ | ✓ | × | × |
| Outlier Suppression | ✓ | ✓ | ✓ | - | ✓ | ✓ |
| PB-LLM | ✓ | ✓ | - | - | ✓ | ✓ |
| BiLLM | ✓ | ✓ | - | - | ✓ | × |
| BitNet | ✓ | ✓ | - | - | - | × |

4. Inference Optimization

Inference optimization primarily involves enhancing the optimization of the forward process of the model, rather than merely modifying the model’s weights or structure. Current large language models generally employ autoregressive decoding based on the decoder-only transformer architecture. Inference optimization aims to fully utilize software and hardware resources by optimizing repeated computations in the forward process, enhancing the efficiency of attention and linear layer operations, and even modifying the model decoding strategy. The main methods for inference optimization include speculative decoding, KV cache optimization, early exiting, kernel optimization, and memory offloading.

4.1. Speculative Decoding

The autoregressive decoding mechanism results in inefficiencies in the generation capabilities of LLMs. Speculative decoding is a type of non-autoregressive decoding algorithm that divides the LLM decoding process into two stages: Drafting and Verifying. First, candidate tokens are generated by a drafting model. Then, the verifying model checks the correctness of candidate tokens and corrects them through a single forward propagation. Typically, the drafting model is smaller, while the verifying model is larger. We give a brief comparison of autoregressive decoding and speculative decoding in Figure 7. The efficiency improve-

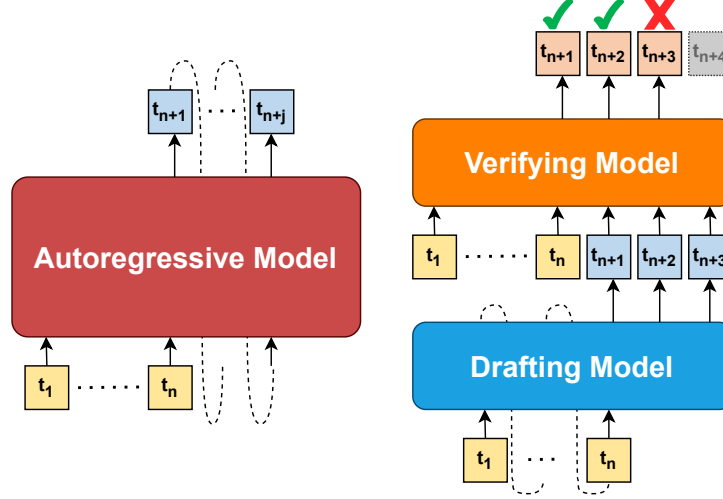


Figure 7: Illustration of autoregressive decoding (left) and speculative decoding (right).

ment in speculative decoding primarily stems from employing auxiliary models with lower resource requirements to generate candidate tokens. Verifying model only needs to parallelly judge the correctness, thereby reducing the end-to-end inference latency. The more candidate tokens accepted by the drafting model and the faster the verifying process, the higher the efficiency of the algorithm. Therefore, efficient drafting and verifying are key research points of speculative decoding.

Studies [139] and [140] first proposed Speculative Sampling, which uses existing pre-trained models of different sizes and a modified sampling schema to achieve accurate decoding, resulting in more than a $2\times$ inference acceleration. BiLD [141] proposed a fallback strategy and rollback strategy based on the prediction distribution to address the issue of when to stop the prediction of a small model and how to verify and correct the predicted tokens. SpecInfer [142] proposed a tree-based decoding mechanism and token tree verification mechanism, reducing memory accesses to parameters and the end-to-end inference latency. EAGLE [143] predicts the next feature at the feature level, uses a trained lightweight autoregressive head to generate tokens, and achieves higher efficiency through tree attention. Medusa [144] adopts multiple decoding heads on top of the last hidden state of an LLM, as well as tree attention, and predicts multiple subsequent tokens in parallel without the need for auxiliary models. Lookahead [145] views autoregressive decoding as solving a nonlinear equation and uses Jacobi iteration method for parallel decoding.

In addition to the above studies, some research has begun applying this tech-

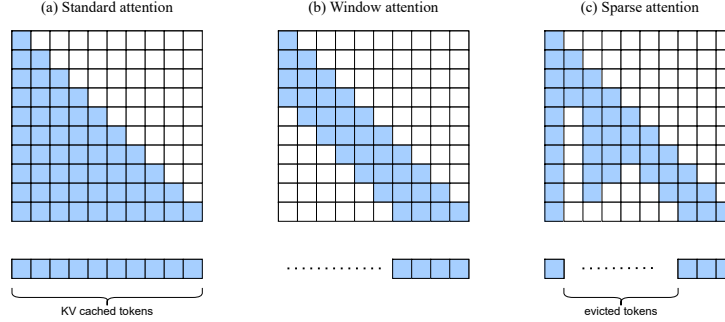


Figure 8: Illustration of KV cache sparsity. (a) shows the full attention and key-value cache, while (b) and (c) depict the sparse attention patterns and corresponding key-value cache.

nology to edge devices to alleviate the memory and computational pressure of LLMs on these devices LLMcad [146] designed a speculative decoding engine based on tree attention and a dynamic fallback strategy, significantly improving the generation speed of LLMs on IoT devices and smartphones. SpecExec [147] generates longer accepted token sequences through an improved token tree structure and designs an LLM parameter offloading system to support interactive LLMs on consumer devices.

4.2. KV Cache Compression

KV cache represents a significant optimization technique in the inference process of LLMs, reducing redundant computations during the decoding stage by retaining previous keys and values in attention heads. The size of the KV cache linearly increases with the length of generated tokens, making memory a bottleneck in the inference process for long text generation. Therefore, researching KV cache compression methods is crucial to enabling efficient long-text inference [148]. Current KV cache compression can be classified into KV cache sparsity and KV cache quantization.

KV Cache Sparsity leverages sparse attention mechanisms to retain only the essential keys and values, thereby reducing the size of the KV cache. Figure 8 illustrates the concept of KV cache sparsity in different attention patterns. FastGen [149] observed that attention heads in different layers exhibit different attention structures and applied different compression strategies based on these patterns. StreamingLLM [150] identifies that in autoregressive LLMs, a large amount of attention score is allocated to the initial few tokens, termed an “attention sink”. By preserving the initial tokens’ KV and employing windowed

attention, it enables LLMs trained with a finite attention window to process text of infinite length. Scissrohands [151] proposed the “Persistence of Importance Hypothesis”, indicating that different tokens have varying importance for future predictions. Leveraging this observation, they proposed an inference algorithm using compressed KV cache, reducing memory usage of KV cache up to $5\times$ without degradation on model quality. H2O [152] introduced a KV cache eviction strategy using attention scores, evicting the key-value pair with the lowest cumulative attention score at each decoding step, thus retaining a balance between recent and important tokens.

KV Cache Quantization reduces the memory usage by quantizing keys and values, similar to weight quantization. However, directly quantizing the KV cache to low precision can lead to significant perplexity degradation [148]. Additionally, the streaming nature of KV cache may introduce additional computational overhead in calculating quantization parameters [153]. To address this issue, KIVI [153] suggests that the key cache should be quantized per-channel, while the value cache should be quantized per-token. Based on this conclusion, they proposed a hardware-friendly 2-bit quantization method, reducing the peak memory usage by $2.6\times$ while maintaining inference quality. KVQuant [148] conducted a more detailed evaluation of KV cache quantization performance, proposing a non-uniform ultra-low precision quantization method for KV cache. They calibrated the quantization parameters of the key cache using an offline method and those of the value cache using an online method. QAQ [154] proposes that key caches and value caches exhibit different sensitivities to quantization and designs a non-uniform mixed quantization strategy to preserve the precision of outlier attention values. IntactKV [155] observes that outliers in attention scores occur at specific tokens, referred to as pivot tokens. It proposes a quantization method using offline calibration and supports treating quantization parameters as additional trainable parameters for further calibration.

4.3. Early Exiting

Early exiting is a conditional computation technique that terminates generation early in the decoding process to reduce computation. Figure 9 illustrates the complete and early exiting processes. CALM [156] introduces an early exit classifier in each layer of the decoder, predicting the confidence score of the current layer. Terminate the decoding early based on whether the confidence score exceeds the threshold. ConsistentEE [157] employs a policy network to make early exiting decisions, while also designing a "Memory Layer" metric incorporated into the reward function, allowing for a balance between accuracy and acceleration based

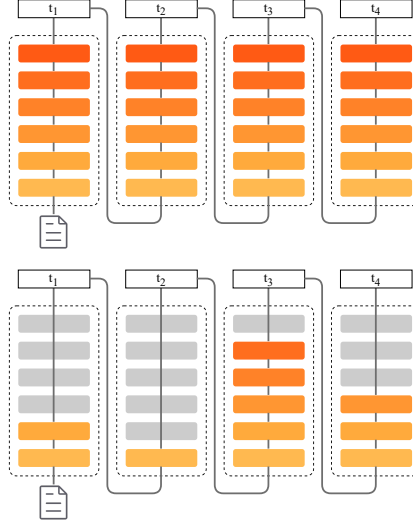


Figure 9: Illustration of complete process (top) and early exiting process (bottom).

on individual hardness. Unfortunately, there are conflicts between early exiting and KV cache, making it difficult for layers after the exit point to accelerate attention calculations [158]. A feasible method is to directly copy the hidden states of the current token at the exiting layer to all subsequent layers. However, this may lead to deviation and degrade output quality [159]. Several recent works aim to mitigate this issue, such as FREE [160], which introduces an early exiting framework incorporating a shallow-deep module and synchronized parallel decoding. It leverages parallel computing capabilities to calculate the KV cache for layers previously early exited when encountering a non-exited token, reducing redundant computations for subsequent tokens. Similarly, EE-LLM [159] proposes a pipeline parallelism relying on multiple devices, thereby filling the KV cache for layers that were early exited. SkipDecode [158] assumes that subsequent tokens are easier to predict, skipping intermediate layers starting from a fixed point. It ensures that the number of skipped layers for preceding tokens is not less than that for subsequent tokens, preventing the recalculation of KV cache. In addition to the above methods, some studies combine early exiting with speculative decoding to further enhance inference efficiency. For instance, LayerSkip [161] employs first few layers of an LLM as a drafting model and the complete model as a verifying model, combining early exiting and speculative decoding strategies. To encourage early exiting, LayerSkip applies layer dropout during training, with the dropout rate increasing as the model depth increases, thereby reducing the

model’s dependency on later layers.

4.4. Kernel Optimization

The main operators of transformers are attention and linear. Optimizing these two operators is crucial for improving the inference efficiency of LLMs. Common optimization methods involve reducing computation and memory access by leveraging software and hardware resources or reducing startup overhead through kernel fusion. FlashAttention [162] proposed an attention tile algorithm that utilizes the high-speed SRAM on GPUs, avoiding communication overhead caused by multiple memory copies through kernel fusion. FlashAttention-2 [163] improves upon the original by optimizing work partitioning among GPU threads, significantly reducing shared memory reads and writes. It further enhances parallelism by distributing attention computation across thread blocks and warps, thereby increasing occupancy and efficiency. FlashDecoding [164] introduces a novel parallelization dimension along the keys/values sequence length, fully utilizing the GPU even with small batch sizes and long contexts. It employs an iterative log-sum-exp reduction to maintain efficient memory usage, significantly reducing attention runtime regardless of sequence length. FlashDecoding++ [165] introduces an asynchronous softmax with a unified max value to eliminate synchronization overheads, significantly enhancing attention computation efficiency. It optimizes flat GEMM operations with double buffering, improving computation utilization and reducing memory latency. Additionally, FlashDecoding++ implements a heuristic dataflow that adapts to hardware resources dynamically, achieving substantial speedups on both NVIDIA and AMD GPUs. PagedAttention [166] leverages an innovative attention algorithm inspired by virtual memory and paging, enabling efficient non-contiguous storage of key-value cache. This method reduces internal and external fragmentation, significantly improving memory utilization. Implemented in the vLLM serving system, PagedAttention achieves 2-4× throughput improvements over state-of-the-art systems by allowing flexible memory management and sharing of KV cache across requests. Flash-LLM [101] proposes a general Load-as-Sparse and Compute-as-Dense methodology for unstructured sparse matrix multiplication (SpMM) to address memory bandwidth bottlenecks in LLM inference. It also includes a pipeline design for SpMM that overlaps sparse data extraction, dense data loading and matrix computation. T-MAC [167] proposed a mixed precision matrix multiplication (mpGEMM) algorithm for weight-quantized LLMs. It reduces multiplications and additions by bit-wise table lookup, achieving higher throughput on lower-end edge devices.

4.5. Memory Offloading

Edge or consumer devices typically have limited memory, making it difficult to load the large parameters of LLMs into the RAM of these devices, rendering LLM inference infeasible. An effective method is to “offload” weights to external storage, loading partial weights into memory when needed, enabling devices to run models larger than their memory capacity. Some platforms and inference engines already provide this feature, such as HuggingFace [168], Llama.cpp [169], etc. However, frequent memory swapping introduces significant communication overhead. To mitigate this drawback, recent studies have optimized memory offloading and loading strategies.

FlexGen [170] developed a linear programming-based search algorithm to optimize throughput within the search space, aiming to identify the optimal offloading strategy. Furthermore, it compresses both weights and attention caches to 4 bits, significantly enhancing the maximum throughput during LLM inference. Chen et al. [171] introduced the concept of attention offloading, which involves partitioning the attention mechanism into multiple subtasks, utilizing memory-optimized devices to compute the attention operator and computation-optimized devices for processing the remaining components of the model. Recent studies have demonstrated significant sparsity in neuron activations during LLM inference [172]. Leveraging this characteristic, some research has designed a memory offloading strategy based on sparsity. PowerInfer [173] discovered that LLM inference exhibits high locality, with some neurons, termed “hot-activated neurons”, being frequently activated. Based on this insight, PowerInfer designed a neuron-aware offloading strategy and inference engine, utilizing both GPU and CPU to store weights. It preloads the weights of frequently activated neurons for the GPU, while the weights of less active neurons are retained on the CPU. Addressing the challenge of designing memory offloading strategies for devices with limited memory, LLM in a flash [174] proposes a memory offloading strategy that utilizes DRAM and flash memory. Specifically, LLM in a flash leverages the concept of contextual sparsity introduced by DeJaVu [172] to dynamically load neurons predicted to be active. It employs a sliding window technique to cache recently activated tokens and utilizes static memory pre-allocation to minimize loading latency. EdgeMoE [175] developed a memory offloading strategy specifically designed for MoE models. Leveraging the sparsity of the MoE architecture, non-expert weights are stored in memory, while expert weights are loaded from external storage only when activated, achieving memory savings on edge devices.

5. Deployment

In this section, we explore frameworks designed for deploying and optimizing LLMs on edge devices. We categorize these frameworks into: on-device inference engines, cloud-edge collaborative frameworks and deployment suites. On-device inference engines refer to software infrastructures optimized for executing LLM inference tasks on edge devices. Cloud-edge collaborative frameworks aims to make full use of the resources of edge devices and the cloud for LLM. Deployment suites are more oriented toward providing software support and enhancing performance for LLM applications. We summarize the main features of these frameworks in Table 5. It is noteworthy that some well-known LLM inference frameworks [166, 176, 177, 178, 179, 180] are primarily utilized for LLM serving. These have not been included in the discussion of this section, despite their capabilities for execution on edge devices.

Table 5: Frameworks designed for deploying and optimizing LLMs on edge devices.

| Category | Features | Works |
|-------------------------------------|---|--|
| On-device Inference Engines | A suite of on-device optimizations, multi-platform compatibility, lightweight inference runtime, convenient development tools, and hardware acceleration capabilities | [181, 182, 183, 184, 146, 173, 185, 186] |
| Cloud-edge Collaborative Frameworks | Combine cloud and edge computing, boost performance for complex tasks, leverage the strengths of both | [187, 188, 189, 190, 191, 192, 193, 194] |
| Deployment Suites | Encompass a variety of LLM inference optimization functionalities and offer convenient development tools | [195, 169, 196, 197] |

5.1. On-device Inference Engines

An inference engine refers to software that combines inference optimization techniques to optimize resources for LLM inference. This category of frameworks includes some general-purpose deep learning inference frameworks for mobile and edge devices, such as ExecuTorch [181], TFLite [182], MNN [183], and NCNN [184]. These engines typically feature a suite of on-device optimizations, multi-platform compatibility, lightweight inference runtime, convenient development tools, and hardware acceleration capabilities. While these engines do

not typically involve specialized optimizations for transformer architectures, their generality and flexibility make them suitable for a wide range of LLM models.

Another category of inference engines includes frameworks specifically designed for LLM inference. These frameworks are optimized for LLM inference tasks, incorporating one or more optimization techniques mentioned in Sec. 4, as well as other on-device optimizations, such as pipeline and efficient kernels. In addition to utilizing tree-based speculative decoding, LLMcad [146] employs a computing-loading pipeline to avoid contention for small and large LLM memory by profiling or user defining a memory upper bound. PowerInfer [173] employs the principle of locality in activations, comprising two stages: offline profiling and online inference. During the online phase, it also designs a sparse matrix multiplication operator tailored for its dynamic scenarios. Based on PowerInfer, PowerInfer-2 [185] introduces a dedicated inference framework for smartphones, leveraging the heterogeneous computational, memory, and I/O resources by decomposing traditional matrix computations into fine-grained computations of neuron clusters. Additionally, it incorporates a fine-grained pipeline to conceal I/O overhead. However, PowerInfer-2 necessitates models to exhibit strong predictable sparsity, which may limit its applicability to current mainstream LLMs. Transformer-Lite [186] is an LLM inference engine specifically designed for smartphones, proposing symbolic expression-based dynamic shape inference, low-precision matrix multiplication operators, and optimization of KV cache copying.

5.2. Cloud-Edge Collaborative Frameworks

Besides the aforementioned on-device LLM inference engines, another strategy is cloud-edge collaborative inference, which combines the advantages of powerful cloud computing and edge-side computation. This approach holds promise as a significant pathway for LLM edge intelligence. For tasks that exceed the capabilities of edge devices, better performance can be achieved through edge-cloud collaboration. Previous studies [187, 188, 189, 190] have employed model partitioning and offloading techniques to distribute computation between the cloud and edge, thereby reducing the computational burden on edge devices. However, due to the autoregressive nature of LLMs, direct model partitioning results in significant communication overhead, making it difficult to apply in LLM applications. Recent studies consider the trade-off between cost and quality for edge-cloud collaboration in resource-constrained scenarios.

Hybrid LLM [191] involves a hybrid inference strategy that considers the trade-off between cost and quality. A router is utilized to allocate queries between a large model (deployed in the cloud) and a small model (suitable for edge

devices). The router’s decision is based on the predicted query difficulty and the desired quality level, which can be dynamically adjusted at test time. This allows for seamless adaptation to different scenarios and requirements. Different router designs, such as deterministic and probabilistic routers with and without data transformation, are explored to handle various situations, especially when dealing with the disparity in performance between the cloud-based large model and the edge-based small model.

Tabi [192] is an efficient multi-level inference system. Tabi features a multi-level inference engine that serves queries using small models and optional LLMs for demanding applications. It decides whether to use small models or LLMs based on calibrated confidence scores. For queries routed to LLMs, it employs attention-based word pruning, using the attention weights from small models to reduce input data size and offset latency overhead. It also uses a weighted ensemble technique to combine predictions from different levels and improve accuracy.

Edge-LLM [193] is a collaborative framework for large language model (LLM) serving in edge computing. The framework aims to address the challenges of efficient deployment and fine-tuning of adapter models for LLM tasks in resource-constrained scenarios. It employs a server-node collaboration approach, where the LLM backbone is deployed on the edge server and the adapter on the edge nodes. This design reduces the computational pressure on the edge server and enables real-time inference and online fine-tuning on the edge nodes.

FedAgg [194] is a framework for End-Edge-Cloud Collaboration in Federated Learning (FL). FedAgg overcomes the model scale limitations of prior Hierarchical Federated Learning (HFL) methods by using a customized Bridge Sample Based Online Distillation Protocol (BSBODP). In BSBODP, each computing node (except leaf nodes with an added encoder) has a pre-trained decoder and a model. Bridge samples, generated from embeddings related to private data, facilitate online distillation between parent - child nodes, enabling knowledge transfer and growth of models in size and generalization ability from end to cloud, while maintaining privacy and flexibility.

Dynamic token-level edge-cloud collaboration framework [198] involves identifying “harder” tokens through various indicators such as Top-1 probabilities or entropy of SLMs and the LLM’s probability distribution on SLM - generated tokens. A draft - verification method is then employed, where tokens with LLM probability above a threshold are from the SLM, and those below are from the LLM. The SLM generates tokens, and the LLM verifies and corrects “harder” tokens, achieving a balance between inference quality and cost.

5.3. *Deployment Suites*

Deployment Suites constitute a category of software infrastructure that is closer to users and developers. Typically, these tools encompass a variety of LLM inference optimization functionalities and offer convenient development tools. MLC-LLM [195] is a general-purpose high-performance LLM local deployment framework that leverages deep learning compilation techniques to provide better optimization and cross-platform portability. MLC-LLM supports a variety of hardware platforms, ranging from smartphones to dedicated edge computing devices, offering great convenience for LLM edge deployment. llama.cpp [169] is a tool for local deployment of LLMs based on pure C++ language. It provides a simple C++ API, supporting the deployment of LLM models on edge devices with various CPU architectures. It also offers optimization techniques, such as model quantization. The core of llama.cpp is GGML [199], an LLM tensor library for edge devices, endowing llama.cpp with strong portability. mllm [196] is a fast and lightweight multimodal LLM inference engine for mobile and edge devices, implemented in C++ and optimized for multiple CPU architectures. It supports various multimodal LLM models and provides a simple API for developers to deploy multimodal LLM models on edge devices. NanoLLM [197] is a local inference framework for LLMs on the NVIDIA Jetson platform. It offers some simple LLM optimization techniques and APIs, providing convenience for LLM inference on the Jetson platform.

6. **Open Challenges and Future Directions**

As LLMs continue to advance and proliferate, their deployment on edge devices presents numerous opportunities and challenges. The potential to bring powerful AI capabilities closer to the user promises significant benefits, including reduced latency, enhanced privacy, and the ability to operate in environments with limited connectivity. However, the transition from cloud-based to edge-based LLM deployment is fraught with obstacles that must be carefully navigated. These challenges range from resource constraints and dynamic edge environments to privacy concerns and ethical considerations. Addressing these issues is critical to fully realizing the potential of LLMs in edge intelligence. Below, we delve into some of the key challenges and future directions in this field.

6.1. *Resource Constraints on Edge Devices*

Edge devices typically lack the performance capabilities of specialized accelerators. Due to the large number of parameters and high demands for compu-

tational and storage resources, LLMs face constraints in terms of storage space, computational power, bandwidth, and energy on edge devices. Traditional lightweight model structures and compression techniques can alleviate this issue to some extent, but they inevitably compromise the performance of LLMs. To address this problem, researchers need to explore more advanced compression and acceleration techniques and design inference engines that fully utilize the hardware of edge devices to improve the efficiency of large model inference in resource-constrained environments. Additionally, designing efficient model architectures is a crucial direction for future research. Recent studies, such as those on MoE [200, 49], RWKV [201], Mamba [202], and RetNet [203], have demonstrated advantages over the classic Transformer [16] architecture. Researching resource-efficient model architectures, particularly those that can adapt to the software and hardware environments of edge devices, can further reduce the deployment and application costs of LLMs.

6.2. *Dynamicity of Edge Environments*

Edge dynamicity encompasses the variability in edge device resources, network conditions, and user demands, posing challenges to the portability and efficiency of deploying and inferring LLMs at the edge. Significant differences in computational and storage resources across devices, coupled with fluctuations in network conditions over time, make it difficult to provide consistent and stable services with LLMs at the edge. Additionally, user requirements for model inference latency and accuracy can vary greatly, further complicating the deployment and inference of LLMs at the edge. Key research directions include modeling the dynamic nature of edge scenarios and developing scheduling mechanisms, collaboration strategies, and adaptive algorithms that can adjust to changes in resources and demands. Notably, in certain LLM deployment scenarios, such as on smartphones, models cannot fully monopolize device resources and may even have lower scheduling priorities. Research on LLMs as system services [204] has already begun and represents an important future direction.

6.3. *Heterogeneity of Edge Devices*

The heterogeneity of edge devices, including differences in hardware configurations, functionalities, and operating systems, presents compatibility challenges for the acceleration and deployment of LLMs. While some progress [205, 206, 207, 208] has been made in serving LLMs on heterogeneous computing platforms, the complexity of heterogeneity in edge environments is even greater. This is particularly true for different hardware such as SoCs, FPGAs, ASICs, and TPUs,

where significant differences in performance and instruction sets pose open questions regarding the portability of LLMs across various platforms. Therefore, it is necessary to model a unified LLM computation paradigm that provides operator compatibility across different platforms. Additionally, exploring adaptive algorithms and flexible deployment frameworks is essential to accommodate diverse edge hardware. Hardware-aware model compression and acceleration techniques can also help optimize on-device inference performance.

6.4. *Edge-Cloud Collaboration*

For tasks that exceed the capabilities of edge devices, better performance can be achieved through edge-cloud collaboration. Previous studies [187, 188, 189, 190] have employed model partitioning and offloading techniques to distribute computation between the cloud and edge, thereby reducing the computational burden on edge devices. However, due to the autoregressive nature of LLMs, direct model partitioning results in significant communication overhead, making it difficult to apply in LLM applications. Another approach is task allocation based on task difficulty, known as Query Routing. Recent studies [191, 192] have designed hybrid inference systems using this method. Designing hybrid inference systems that adaptively allocate queries of varying difficulty between the cloud and edge is a promising research direction.

6.5. *Privacy and Security Concerns*

Deploying LLMs on edge devices raises significant privacy and security concerns. Sensitive data processed on edge devices can be vulnerable to breaches, making it essential to develop robust encryption methods, secure model execution environments, and privacy-preserving machine learning techniques. Ensuring data privacy while maintaining model performance is a critical challenge. Additionally, securing the models themselves against adversarial attacks and unauthorized access is paramount to protect both user data and the integrity of the models.

6.6. *Hallucination Issues*

Gunasekar et al. [12] suggests that high-quality data can enable even small-sized LLMs to achieve high performance, inspiring many subsequent studies. However, small language models, due to their limited number of parameters, often exhibit significant performance variability, with hallucination issues being particularly prominent. Recent study [209] indicate that small language models suffer from severe overfitting problems, which hinder their application at the edge.

Exploring techniques to mitigate overfitting and reduce hallucination in small language models is crucial for enhancing their reliability and usability in edge scenarios. Potential solutions may involve advanced regularization techniques, improved training methods, and hybrid approaches that combine the strengths of both small and large models.

6.7. Morality and law

As the expansion of LLMs to edge environments becomes increasingly prevalent, the significance of ethical and legal challenges becomes more pronounced. We anticipate that the behavior of LLMs aligns with human values, objectives, and expectations, ensuring their adherence to ethical standards and legal regulations, especially in sensitive domains such as healthcare and finance. However, deploying LLMs on edge devices typically requires techniques such as pruning and distillation to reduce the model's size and computational demands. These methods, however, may inadvertently impair the model's ability to adhere to ethical guidelines, potentially leading to biased or harmful outputs. Furthermore, the decentralized nature of edge computing complicates the attribution of responsibility. Determining responsibility becomes extremely challenging when LLMs generate inappropriate content or violate privacy regulations.

6.8. Interpretability

As LLMs become increasingly complex, comprehending their decision-making processes becomes more challenging. The obscurity of their internal mechanisms and lack of transparency introduce unnecessary risks to downstream applications, particularly in critical domains such as healthcare where a deep understanding of how decisions are made is essential. To deploy large models on edge devices, optimization techniques such as pruning or distillation are commonly employed. While these methods effectively reduce the model's size and computational requirements, they may lead to a simplification of the model's structure, thereby diminishing its interpretability. The reduction of intermediate layers and complexity could result in the loss of information crucial for explaining the model's behavior. Consequently, ensuring the interpretability of models while maintaining performance on edge devices is one of the pressing challenges that future research must address.

6.9. User preferences and personalization

The deployment of LLMs on edge devices naturally increases the demand for personalized interactions due to their proximity to users and the environments in

which they operate. Additionally, edge devices can collect a wealth of personalized data, such as location, device usage patterns, and user behavior, enabling models to tailor responses more effectively to individual needs. Research has already begun to focus on the personalization of large models [210], Doddapaneni et al. [211] introduced a User Embedding Model (UEM) that compresses user history into embedding vectors as soft prompts for language models, significantly enhancing the personalization accuracy of recommendation systems. Developing the capability for LLMs to learn and adapt in real-time from user interactions, ensuring that models can adjust to personal preferences while safeguarding data privacy, is a promising avenue for future research. By incorporating controllable mechanisms for knowledge retention and forgetting, as well as continuous learning mechanisms, models can autonomously learn new skills and improve existing capabilities based on user interactions and local data. With this capability, edge LLMs can leverage on-device learning for personalized content delivery and enhanced user experiences, while also minimizing latency.

6.10. Green and sustainable development

Training and processing of LLMs require substantial data and significant computational resources. Consequently, LLMs are typically trained and deployed on resource-rich cloud servers. However, as large models increasingly migrate to wireless edge networks closer to end-user devices, edge AI faces considerable challenges. Operating resource-intensive models on battery-powered devices can rapidly deplete battery life. STI [212] employs an elastic pipeline planning with a preloaded buffer, effectively balancing latency and memory usage on mobile devices, thereby accelerating the NLP inference process on edge devices. This acceleration maximizes the utilization of computational resources. Recently, Yuan et al. [213] have proposed a novel mobile AI paradigm that integrates a foundational model on the NPU of mobile devices to achieve unified management and optimization for a variety of mobile AI tasks, and enhances efficiency and performance by providing short, offline fine-tuned adapters for different applications.

7. Conclusion

We have provided a comprehensive overview of the current state of LLMs and their deployment on edge devices, highlighting various techniques and strategies to overcome inherent challenges. We explored the development of small language models, model compression techniques, inference optimization strategies, and frameworks designed for efficient edge deployment. Despite significant

advancements, many challenges remain, presenting formidable obstacles to the widespread adoption of LLMs at the edge. We hope our survey has shed light on these challenges and provided valuable insights into edge intelligence enabled by LLMs for researchers and practitioners alike.

References

- [1] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, Improving Language Understanding by Generative Pre-Training, Technical Report, OpenAI, 2018.
- [2] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, in: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Association for Computational Linguistics, Minneapolis, Minnesota, 2019, pp. 4171–4186.
- [3] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, et al., GPT-4 Technical Report, 2024. [arXiv:2303.08774](https://arxiv.org/abs/2303.08774).
- [4] B. Workshop, T. L. Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilić, D. Hesslow, R. Castagné, A. S. Luccioni, F. Yvon, M. Gallé, J. Tow, A. M. Rush, S. Biderman, A. Webson, P. S. Ammanamanchi, T. Wang, B. Sagot, N. Muenighoff, A. V. del Moral, et al., BLOOM: A 176B-Parameter Open-Access Multilingual Language Model, 2023. [arXiv:2211.05100](https://arxiv.org/abs/2211.05100).
- [5] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, G. Lample, LLaMA: Open and Efficient Foundation Language Models, 2023. [arXiv:2302.13971](https://arxiv.org/abs/2302.13971).
- [6] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, et al., Llama 2: Open Foundation and Fine-Tuned Chat Models, 2023. [arXiv:2307.09288](https://arxiv.org/abs/2307.09288).

- [7] N. Dhar, B. Deng, D. Lo, X. Wu, L. Zhao, K. Suo, An Empirical Analysis and Resource Footprint Study of Deploying Large Language Models on Edge Devices, in: Proceedings of the 2024 ACM Southeast Conference, ACM SE '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 69–76.
- [8] X. Li, Z. Lu, D. Cai, X. Ma, M. Xu, Large Language Models on Mobile Devices: Measurements, Analysis, and Insights, in: Proceedings of the Workshop on Edge and Mobile Foundation Models, EdgeFM '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 1–6.
- [9] M. Xu, H. Du, D. Niyato, J. Kang, Z. Xiong, S. Mao, Z. Han, A. Jamalipour, D. I. Kim, X. Shen, V. C. M. Leung, H. V. Poor, Unleashing the Power of Edge-Cloud Generative AI in Mobile Networks: A Survey of AIGC Services, IEEE Communications Surveys & Tutorials 26 (2024) 1127–1170.
- [10] L. S. Vailshery, IoT connections worldwide 2022-2033, ??? URL: <https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/>, (Accessed: Jul. 26, 2024).
- [11] C. Fourrier, N. Habib, A. Lozovskaya, K. Szafer, T. Wolf, Open llm leaderboard v2, 2024. URL: https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard.
- [12] S. Gunasekar, Y. Zhang, J. Aneja, C. C. T. Mendes, A. Del Giorno, S. Gopi, M. Javaheripi, P. Kauffmann, G. de Rosa, O. Saarikivi, A. Salim, S. Shah, H. S. Behl, X. Wang, S. Bubeck, R. Eldan, A. T. Kalai, Y. T. Lee, Y. Li, Textbooks Are All You Need, 2023. [arXiv:2306.11644](https://arxiv.org/abs/2306.11644).
- [13] S. Hu, Y. Tu, X. Han, C. He, G. Cui, X. Long, Z. Zheng, Y. Fang, Y. Huang, W. Zhao, X. Zhang, Z. L. Thai, K. Zhang, C. Wang, Y. Yao, C. Zhao, J. Zhou, J. Cai, Z. Zhai, N. Ding, et al., MiniCPM: Unveiling the Potential of Small Language Models with Scalable Training Strategies, 2024. [arXiv:2404.06395](https://arxiv.org/abs/2404.06395).
- [14] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebron, S. Sanghai, GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints, in: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Singapore, 2023, pp. 4895–4901.

- [15] N. Shazeer, Fast transformer decoding: One write-head is all you need, 2019. [arXiv:1911.02150](https://arxiv.org/abs/1911.02150).
- [16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is All you Need, in: *Advances in Neural Information Processing Systems*, volume 30, Curran Associates, Inc., 2017.
- [17] O. Press, L. Wolf, Using the output embedding to improve language models, in: M. Lapata, P. Blunsom, A. Koller (Eds.), *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, Association for Computational Linguistics, Valencia, Spain, 2017, pp. 157–163. URL: <https://aclanthology.org/E17-2025>.
- [18] Z. Liu, C. Zhao, F. Iandola, C. Lai, Y. Tian, I. Fedorov, Y. Xiong, E. Chang, Y. Shi, R. Krishnamoorthi, L. Lai, V. Chandra, MobileLLM: Optimizing Sub-billion Parameter Language Models for On-Device Use Cases, 2024. [arXiv:2402.14905](https://arxiv.org/abs/2402.14905).
- [19] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, W. Fedus, Emergent abilities of large language models, *Transactions on Machine Learning Research* (2022). URL: <https://openreview.net/forum?id=yzkSU5zdwD>.
- [20] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, RoBERTa: A Robustly Optimized BERT Pre-training Approach, 2019. [arXiv:1907.11692](https://arxiv.org/abs/1907.11692).
- [21] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, C. Leahy, The Pile: An 800GB Dataset of Diverse Text for Language Modeling, 2020. [arXiv:2101.00027](https://arxiv.org/abs/2101.00027).
- [22] E. M. Bender, T. Gebru, A. McMillan-Major, S. Shmitchell, On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?, in: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency, FAccT '21*, Association for Computing Machinery, New York, NY, USA, 2021, pp. 610–623.

- [23] H. Laurençon, L. Saulnier, T. Wang, C. Akiki, A. V. del Moral, T. Le Scao, L. von Werra, C. Mou, E. G. Ponferrada, H. Nguyen, J. Frohberg, M. Šaško, Q. Lhoest, A. McMillan-Major, G. Dupont, S. Biderman, A. Rogers, L. Benallal, F. De Toni, G. Pistilli, et al., The BigScience ROOTS corpus: a 1.6TB composite multilingual dataset, in: Proceedings of the 36th International Conference on Neural Information Processing Systems, NIPS '22, Curran Associates Inc., Red Hook, NY, USA, 2024, pp. 31809–31826.
- [24] D. Soboleva, SlimPajama: A 627B token, cleaned and deduplicated version of RedPajama, ??? URL: <https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama/>, (Accessed: Jul. 26, 2024).
- [25] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim, Q. Liu, E. Zheltonozhskii, T. Y. Zhuo, T. Wang, O. Dehaene, J. Lamy-Poirier, J. Monteiro, N. Gontier, M.-H. Yee, L. K. Umapathi, et al., StarCoder: may the source be with you!, Transactions on Machine Learning Research (2023).
- [26] M. Bellagente, J. Tow, D. Mahan, D. Phung, M. Zhuravinskiy, R. Adithyan, J. Baicoianu, B. Brooks, N. Cooper, A. Datta, M. Lee, E. Mostaque, M. Pieler, N. Pinnaparju, P. Rocha, H. Saini, H. Teufel, N. Zanichelli, C. Riquelme, Stable LM 2 1.6B Technical Report, 2024. [arXiv:2402.17834](https://arxiv.org/abs/2402.17834).
- [27] G. Penedo, Q. Malartic, D. Hesslow, R. Cojocar, H. Alobeidli, A. Cappelli, B. Pannier, E. Almazrouei, J. Launay, The RefinedWeb Dataset for Falcon LLM: Outperforming Curated Corpora with Web Data Only, Advances in Neural Information Processing Systems 36 (2023) 79155–79172.
- [28] T. Computer, RedPajama: an open dataset for training large language models, ??? URL: <https://github.com/togethercomputer/RedPajama-Data>, (Accessed: Mar. 30, 2024).
- [29] L. Soldaini, R. Kinney, A. Bhagia, D. Schwenk, D. Atkinson, R. Author, B. Bogin, K. Chandu, J. Dumas, Y. Elazar, V. Hofmann, A. H. Jha, S. Kumar, L. Lucy, X. Lyu, N. Lambert, I. Magnusson, J. Morrison, N. Muennighoff, A. Naik, et al., Dolma: an Open Corpus of Three Trillion Tokens for Language Model Pretraining Research, 2024. [arXiv:2402.00159](https://arxiv.org/abs/2402.00159).

- [30] Common Crawl - Open Repository of Web Crawl Data, ??? URL: <https://commoncrawl.org/>, (Accessed: Jul. 26, 2024).
- [31] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer, *Journal of Machine Learning Research* 21 (2020) 1–67.
- [32] S. Yuan, H. Zhao, Z. Du, M. Ding, X. Liu, Y. Cen, X. Zou, Z. Yang, J. Tang, WuDaoCorpora: A super large-scale Chinese corpora for pre-training language models, *AI Open* 2 (2021) 65–68.
- [33] FudanNLPLAB/CBook-150K, ??? URL: <https://github.com/FudanNLPLAB/CBook-150K>, (Accessed: Jul. 26, 2024).
- [34] L. B. Allal, A. Lozhkov, E. Bakouch, L. von Werra, T. Wolf, Smollm - blazingly fast and remarkably powerful, 2024. URL: <https://huggingface.co/blog/smollm>.
- [35] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, L. Zettlemoyer, OPT: Open Pre-trained Transformer Language Models, 2022. [arXiv:2205.01068](https://arxiv.org/abs/2205.01068).
- [36] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, et al., Language Models are Few-Shot Learners, in: *Advances in Neural Information Processing Systems*, volume 33, Curran Associates, Inc., 2020, pp. 1877–1901.
- [37] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, H. Wu, Mixed Precision Training, 2018. [arXiv:1710.03740](https://arxiv.org/abs/1710.03740).
- [38] Z. Du, Y. Qian, X. Liu, M. Ding, J. Qiu, Z. Yang, J. Tang, GLM: General Language Model Pretraining with Autoregressive Blank Infilling, in: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Association for Computational Linguistics, Dublin, Ireland, 2022, pp. 320–335.

- [39] T. GLM, A. Zeng, B. Xu, B. Wang, C. Zhang, D. Yin, D. Rojas, G. Feng, H. Zhao, H. Lai, H. Yu, H. Wang, J. Sun, J. Zhang, J. Cheng, J. Gui, J. Tang, J. Zhang, J. Li, L. Zhao, et al., ChatGLM: A Family of Large Language Models from GLM-130B to GLM-4 All Tools, 2024. [arXiv:2406.12793](https://arxiv.org/abs/2406.12793).
- [40] Z. Cai, M. Cao, H. Chen, K. Chen, K. Chen, X. Chen, X. Chen, Z. Chen, Z. Chen, P. Chu, X. Dong, H. Duan, Q. Fan, Z. Fei, Y. Gao, J. Ge, C. Gu, Y. Gu, T. Gui, A. Guo, et al., InternLM2 Technical Report, 2024. [arXiv:2403.17297](https://arxiv.org/abs/2403.17297).
- [41] InternLM, Internlm/internlm2_5-1_8b, 2024. URL: https://huggingface.co/internlm/internlm2_5-1_8b.
- [42] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang, B. Hui, L. Ji, M. Li, J. Lin, R. Lin, D. Liu, G. Liu, C. Lu, K. Lu, J. Ma, et al., Qwen Technical Report, 2023. [arXiv:2309.16609](https://arxiv.org/abs/2309.16609).
- [43] Introducing Qwen1.5, ??? URL: <http://qwenlm.github.io/blog/qwen1.5/>, (Accessed: Jun. 18, 2024).
- [44] A. Yang, B. Yang, B. Hui, B. Zheng, B. Yu, C. Zhou, C. Li, C. Li, D. Liu, F. Huang, G. Dong, H. Wei, H. Lin, J. Tang, J. Wang, J. Yang, J. Tu, J. Zhang, J. Ma, J. Yang, J. Xu, J. Zhou, J. Bai, J. He, J. Lin, K. Dang, K. Lu, K. Chen, K. Yang, M. Li, M. Xue, N. Ni, P. Zhang, P. Wang, R. Peng, R. Men, R. Gao, R. Lin, S. Wang, S. Bai, S. Tan, T. Zhu, T. Li, T. Liu, W. Ge, X. Deng, X. Zhou, X. Ren, X. Zhang, X. Wei, X. Ren, X. Liu, Y. Fan, Y. Yao, Y. Zhang, Y. Wan, Y. Chu, Y. Liu, Z. Cui, Z. Zhang, Z. Guo, Z. Fan, Qwen2 technical report, 2024. URL: <http://arxiv.org/abs/2407.10671>. doi:10.48550/arXiv.2407.10671. [arXiv:2407.10671](https://arxiv.org/abs/2407.10671).
- [45] Qwen Team, Qwen2.5: A party of foundation models!, 2024. URL: <http://qwenlm.github.io/blog/qwen2.5/>.
- [46] Y. Li, S. Bubeck, R. Eldan, A. Del Giorno, S. Gunasekar, Y. T. Lee, Textbooks Are All You Need II: phi-1.5 technical report, 2023. [arXiv:2309.05463](https://arxiv.org/abs/2309.05463).
- [47] A. Hughes, Phi-2: The surprising power of small language models, ??? URL: <https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/>, (Accessed: Jun. 18 2024).

- [48] M. Abdin, S. A. Jacobs, A. A. Awan, J. Aneja, A. Awadallah, H. Awadalla, N. Bach, A. Bahree, A. Bakhtiari, J. Bao, H. Behl, A. Benhaim, M. Bilenko, J. Bjorck, S. Bubeck, Q. Cai, M. Cai, C. C. T. Mendes, W. Chen, V. Chaudhary, et al., Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone, 2024. [arXiv:2404.14219](https://arxiv.org/abs/2404.14219).
- [49] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. B. Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. R. Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, et al., Mixtral of Experts, 2024. [arXiv:2401.04088](https://arxiv.org/abs/2401.04088).
- [50] G. Team, T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, S. Pathak, L. Sifre, M. Rivière, M. S. Kale, J. Love, P. Tafti, L. Hussenot, P. G. Sessa, A. Chowdhery, A. Roberts, A. Barua, A. Botev, A. Castro-Ros, A. Slone, A. Héliou, et al., Gemma: Open Models Based on Gemini Research and Technology, 2024. [arXiv:2403.08295](https://arxiv.org/abs/2403.08295).
- [51] P. Zhang, G. Zeng, T. Wang, W. Lu, TinyLlama: An Open-Source Small Language Model, 2024. [arXiv:2401.02385](https://arxiv.org/abs/2401.02385).
- [52] S. Biderman, H. Schoelkopf, Q. G. Anthony, H. Bradley, K. O’Brien, E. Hallahan, M. A. Khan, S. Purohit, U. S. Prashanth, E. Raff, A. Skowron, L. Sutawika, O. V. D. Wal, Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling, in: Proceedings of the 40th International Conference on Machine Learning, PMLR, 2023, pp. 2397–2430.
- [53] O. Thawakar, A. Vayani, S. Khan, H. Cholakal, R. M. Anwer, M. Felsberg, T. Baldwin, E. P. Xing, F. S. Khan, MobiLlama: Towards Accurate and Lightweight Fully Transparent GPT, 2024. [arXiv:2402.16840](https://arxiv.org/abs/2402.16840).
- [54] D. Groeneveld, I. Beltagy, P. Walsh, A. Bhagia, R. Kinney, O. Tafjord, A. H. Jha, H. Ivison, I. Magnusson, Y. Wang, S. Arora, D. Atkinson, R. Authur, K. R. Chandu, A. Cohan, J. Dumas, Y. Elazar, Y. Gu, J. Hessel, T. Khot, et al., OLMo: Accelerating the Science of Language Models, 2024. [arXiv:2402.00838](https://arxiv.org/abs/2402.00838).
- [55] OpenBMB, Openbmb/minicpm3-4b, 2024. URL: <https://huggingface.co/openbmb/MiniCPM3-4B>.

- [56] DeepSeek-AI, A. Liu, Bin Wang, B. Wang, B. Wang, B. Liu, C. Zhao, C. Dengr, C. Ruan, D. Dai, D. Guo, D. Yang, D. Chen, D. Ji, E. Li, F. Lin, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Xu, H. Yang, H. Zhang, H. Ding, H. Xin, H. Gao, H. Li, H. Qu, J. L. Cai, J. Liang, J. Guo, J. Ni, J. Li, J. Chen, J. Yuan, J. Qiu, J. Song, K. Dong, K. Gao, K. Guan, L. Wang, L. Zhang, L. Xu, L. Xia, L. Zhao, L. Zhang, M. Li, M. Wang, M. Zhang, M. Zhang, M. Tang, M. Li, N. Tian, P. Huang, P. Wang, P. Zhang, Q. Zhu, Q. Chen, Q. Du, R. J. Chen, R. L. Jin, R. Ge, R. Pan, R. Xu, R. Chen, S. S. Li, S. Lu, S. Zhou, S. Chen, S. Wu, S. Ye, S. Ma, S. Wang, S. Zhou, S. Yu, S. Zhou, S. Zheng, T. Wang, T. Pei, T. Yuan, T. Sun, W. L. Xiao, W. Zeng, W. An, W. Liu, W. Liang, W. Gao, W. Zhang, X. Q. Li, X. Jin, X. Wang, X. Bi, X. Liu, X. Wang, X. Shen, X. Chen, X. Chen, X. Nie, X. Sun, X. Wang, X. Liu, X. Xie, X. Yu, X. Song, X. Zhou, X. Yang, X. Lu, X. Su, Y. Wu, Y. K. Li, Y. X. Wei, Y. X. Zhu, Y. Xu, Y. Huang, Y. Li, Y. Zhao, Y. Sun, Y. Li, Y. Wang, Y. Zheng, Y. Zhang, Y. Xiong, Y. Zhao, Y. He, Y. Tang, Y. Piao, Y. Dong, Y. Tan, Y. Liu, Y. Wang, Y. Guo, Y. Zhu, Y. Wang, Y. Zou, Y. Zha, Y. Ma, Y. Yan, Y. You, Y. Liu, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Huang, Z. Zhang, Z. Xie, Z. Hao, Z. Shao, Z. Wen, Z. Xu, Z. Zhang, Z. Li, Z. Wang, Z. Gu, Z. Li, Z. Xie, Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024. URL: <http://arxiv.org/abs/2405.04434>. doi:10.48550/arXiv.2405.04434. arXiv:2405.04434.
- [57] S. Mehta, M. H. Sekhavat, Q. Cao, M. Horton, Y. Jin, C. Sun, I. Mirzadeh, M. Najibi, D. Belenko, P. Zatloukal, M. Rastegari, OpenELM: An Efficient Language Model Family with Open Training and Inference Framework, 2024. [arXiv:2404.14619](https://arxiv.org/abs/2404.14619).
- [58] B. Zhang, R. Sennrich, Root Mean Square Layer Normalization, in: Advances in Neural Information Processing Systems, volume 32, Curran Associates, Inc., 2019.
- [59] L. B. Allal, A. Lozhkov, E. Bakouch, L. von Werra, T. Wolf, Smollm2 - with great data, comes great performance, 2024. URL: <https://huggingface.co/HuggingFaceTB/SmolLM2-1.7B>.
- [60] P. Singer, P. Pfeiffer, Y. Babakhin, M. Jeblick, N. Dhankhar, G. Fodor, S. S. Ambati, H2o-danube-1.8b technical report, 2024. URL: [http:](http://arxiv.org/abs/2405.04434)

[//arxiv.org/abs/2401.16818](https://arxiv.org/abs/2401.16818). doi:10.48550/arXiv.2401.16818.
arXiv:2401.16818.

- [61] P. Pfeiffer, P. Singer, Y. Babakhin, G. Fodor, N. Dhankhar, S. S. Ambati, H2o-danube3 technical report, 2024. URL: <http://arxiv.org/abs/2407.09276>. doi:10.48550/arXiv.2407.09276. arXiv:2407.09276.
- [62] G. Team, S. Borgeaud, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, K. Millican, D. Silver, M. Johnson, I. Antonoglou, J. Schrittwieser, A. Glaese, J. Chen, E. Pitler, T. Lillicrap, A. Lazaridou, O. Firat, J. Molloy, et al., Gemini: A Family of Highly Capable Multimodal Models, 2024. arXiv:2312.11805.
- [63] M. Wu, A. Waheed, C. Zhang, M. Abdul-Mageed, A. F. Aji, LaMini-LM: A Diverse Herd of Distilled Models from Large-Scale Instructions, in: Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, St. Julian’s, Malta, 2024, pp. 944–964.
- [64] I. Timiryasov, J.-L. Tastet, Baby Llama: knowledge distillation from an ensemble of teachers trained on a small dataset with no performance penalty, 2023. arXiv:2308.02019.
- [65] A. Warstadt, A. Mueller, L. Choshen, E. Wilcox, C. Zhuang, J. Ciro, R. Mosquera, B. Paranjabe, A. Williams, T. Linzen, R. Cotterell, Findings of the BabyLM Challenge: Sample-Efficient Pretraining on Developmentally Plausible Corpora, in: Proceedings of the BabyLM Challenge at the 27th Conference on Computational Natural Language Learning, Association for Computational Linguistics, Singapore, 2023, pp. 1–34.
- [66] Y. Gu, L. Dong, F. Wei, M. Huang, MiniLLM: Knowledge Distillation of Large Language Models, in: The Twelfth International Conference on Learning Representations, 2023.
- [67] C. Zhang, D. Song, Z. Ye, Y. Gao, Towards the Law of Capacity Gap in Distilling Language Models, 2024. arXiv:2311.07052.
- [68] M. Xia, T. Gao, Z. Zeng, D. Chen, Sheared llama: Accelerating language model pre-training via structured pruning, in: The Twelfth International Conference on Learning Representations, 2023. URL: <https://openreview.net/forum?id=09i0dae0zp>.

- [69] Meta, Llama 3.2: Revolutionizing edge ai and vision with open, customizable models, 2024. URL: <https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/>.
- [70] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. K. Li, F. Luo, Y. Xiong, W. Liang, DeepSeek-Coder: When the Large Language Model Meets Programming – The Rise of Code Intelligence, 2024. [arXiv:2401.14196](#).
- [71] C. Team, H. Zhao, J. Hui, J. Howland, N. Nguyen, S. Zuo, A. Hu, C. A. Choquette-Choo, J. Shen, J. Kelley, K. Bansal, L. Vilnis, M. Wirth, P. Michel, P. Choy, P. Joshi, R. Kumar, S. Hashmi, S. Agrawal, Z. Gong, J. Fine, T. Warkentin, A. J. Hartman, B. Ni, K. Korevec, K. Schaefer, S. Huffman, CodeGemma: Open Code Models Based on Gemma, 2024. [arXiv:2406.11409](#).
- [72] A. Mitra, H. Khanpour, C. Rosset, A. Awadallah, Orca-Math: Unlocking the potential of SLMs in Grade School Math, 2024. [arXiv:2402.14830](#).
- [73] Y. Zhu, M. Zhu, N. Liu, Z. Ou, X. Mou, J. Tang, LLaVA-Phi: Efficient Multi-Modal Assistant with Small Language Model, 2024. [arXiv:2401.02330](#).
- [74] Y. Yang, H. Sun, J. Li, R. Liu, Y. Li, Y. Liu, H. Huang, Y. Gao, MindLLM: Pre-training Lightweight Large Language Model from Scratch, Evaluations and Domain Applications, 2023. [arXiv:2310.15777](#).
- [75] X. Chu, L. Qiao, X. Lin, S. Xu, Y. Yang, Y. Hu, F. Wei, X. Zhang, B. Zhang, X. Wei, C. Shen, MobileVLM : A Fast, Strong and Open Vision Language Assistant for Mobile Devices, 2023. [arXiv:2312.16886](#).
- [76] X. Chu, L. Qiao, X. Zhang, S. Xu, F. Wei, Y. Yang, X. Sun, Y. Hu, X. Lin, B. Zhang, C. Shen, MobileVLM V2: Faster and Stronger Baseline for Vision Language Model, 2024. [arXiv:2402.03766](#).
- [77] B. Zhou, Y. Hu, X. Weng, J. Jia, J. Luo, X. Liu, J. Wu, L. Huang, TinyLLaVA: A Framework of Small-scale Large Multimodal Models, 2024. [arXiv:2402.14289](#).
- [78] M. He, Y. Liu, B. Wu, J. Yuan, Y. Wang, T. Huang, B. Zhao, Efficient Multi-modal Learning from Data-centric Perspective, 2024. [arXiv:2402.11530](#).

- [79] H. Lu, W. Liu, B. Zhang, B. Wang, K. Dong, B. Liu, J. Sun, T. Ren, Z. Li, H. Yang, Y. Sun, C. Deng, H. Xu, Z. Xie, C. Ruan, DeepSeek-VL: Towards Real-World Vision-Language Understanding, 2024. [arXiv:2403.05525](https://arxiv.org/abs/2403.05525).
- [80] H. Wei, L. Kong, J. Chen, L. Zhao, Z. Ge, E. Yu, J. Sun, C. Han, X. Zhang, Small Language Model Meets with Reinforced Vision Vocabulary, 2024. [arXiv:2401.12503](https://arxiv.org/abs/2401.12503).
- [81] Y. Yao, T. Yu, A. Zhang, C. Wang, J. Cui, H. Zhu, T. Cai, H. Li, W. Zhao, Z. He, Q. Chen, H. Zhou, Z. Zou, H. Zhang, S. Hu, Z. Zheng, J. Zhou, J. Cai, X. Han, G. Zeng, D. Li, Z. Liu, M. Sun, Minicpm-v: A gpt-4v level mllm on your phone, 2024. URL: <http://arxiv.org/abs/2408.01800>. doi:10.48550/arXiv.2408.01800. [arXiv:2408.01800](https://arxiv.org/abs/2408.01800).
- [82] Y. Jin, J. Li, Y. Liu, T. Gu, K. Wu, Z. Jiang, M. He, B. Zhao, X. Tan, Z. Gan, Y. Wang, C. Wang, L. Ma, Efficient Multimodal Large Language Models: A Survey, 2024. [arXiv:2405.10739](https://arxiv.org/abs/2405.10739).
- [83] W. Chen, Z. Li, M. Ma, Octopus: On-device language model for function calling of software apis, 2024. URL: <http://arxiv.org/abs/2404.01549>. doi:10.48550/arXiv.2404.01549. [arXiv:2404.01549](https://arxiv.org/abs/2404.01549).
- [84] W. Chen, Z. Li, Octopus v2: On-device language model for super agent, 2024. URL: <http://arxiv.org/abs/2404.01744>. doi:10.48550/arXiv.2404.01744. [arXiv:2404.01744](https://arxiv.org/abs/2404.01744).
- [85] W. Chen, Z. Li, Octopus v3: Technical report for on-device sub-billion multimodal ai agent, 2024. URL: <http://arxiv.org/abs/2404.11459>. doi:10.48550/arXiv.2404.11459. [arXiv:2404.11459](https://arxiv.org/abs/2404.11459).
- [86] W. Chen, Z. Li, Octopus v4: Graph of language models, 2024. URL: <http://arxiv.org/abs/2404.19296>. doi:10.48550/arXiv.2404.19296. [arXiv:2404.19296](https://arxiv.org/abs/2404.19296).
- [87] T. Gunter, Z. Wang, C. Wang, R. Pang, A. Narayanan, A. Zhang, B. Zhang, C. Chen, C.-C. Chiu, D. Qiu, D. Gopinath, D. A. Yap, D. Yin, F. Nan, F. Weers, G. Yin, H. Huang, J. Wang, J. Lu, J. Peebles, K. Ye, M. Lee, N. Du, Q. Chen, Q. Keunebroek, S. Wiseman, S. Evans, T. Lei, V. Rathod, X. Kong, X. Du, Y. Li, Y. Wang, Y. Gao, Z. Ahmed, Z. Xu, Z. Lu, A. Rashid, A. M. Jose, A. Doane, A. Bencomo, A. Vanderby, A. Hansen,

- A. Jain, A. M. Anupama, A. Kamal, B. Wu, C. Brum, C. Maalouf, C. Erdenbileg, C. Dulhanty, D. Moritz, D. Kang, E. Jimenez, E. Ladd, F. Shi, F. Bai, F. Chu, F. Hohman, H. Kotek, H. G. Coleman, J. Li, J. Bigham, J. Cao, J. Lai, J. Cheung, J. Shan, J. Zhou, J. Li, J. Qin, K. Singh, K. Vega, K. Zou, L. Heckman, L. Gardiner, M. Bowler, M. Cordell, M. Cao, N. Hay, N. Shahdadpuri, O. Godwin, P. Dighe, P. Rachapudi, R. Tantawi, R. Frigg, S. Davarnia, S. Shah, S. Guha, S. Sirovica, S. Ma, S. Ma, S. Wang, S. Kim, S. Jayaram, V. Shankar, V. Paidi, V. Kumar, X. Wang, X. Zheng, W. Cheng, Y. Shrager, Y. Ye, Y. Tanaka, Y. Guo, Y. Meng, Z. T. Luo, Z. Ouyang, A. Aygar, A. Wan, A. Walkingshaw, A. Narayanan, A. Lin, A. Farooq, B. Ramerth, C. Reed, C. Bartels, C. Chaney, D. Riazati, E. L. Yang, E. Feldman, G. Hochstrasser, G. Seguin, I. Belousova, J. Pelemans, K. Yang, K. A. Vahid, L. Cao, M. Najibi, M. Zuliani, M. Horton, M. Cho, N. Bhendawade, P. Dong, P. Maj, P. Agrawal, Q. Shan, Q. Fu, R. Poston, S. Xu, S. Liu, S. Rao, T. Heeramun, T. Merth, U. Rayala, V. Cui, V. R. Sridhar, W. Zhang, W. Zhang, W. Wu, X. Zhou, X. Liu, Y. Zhao, Y. Xia, Z. Ren, Z. Ren, Apple intelligence foundation language models, 2024. URL: <http://arxiv.org/abs/2407.21075>. doi:10.48550/arXiv.2407.21075. arXiv:2407.21075.
- [88] C. Zhang, Z. Yang, J. Liu, Y. Han, X. Chen, Z. Huang, B. Fu, G. Yu, Appagent: Multimodal agents as smartphone users, arXiv preprint arXiv:2312.13771 (2023).
- [89] A. Yan, Z. Yang, W. Zhu, K. Lin, L. Li, J. Wang, J. Yang, Y. Zhong, J. McAuley, J. Gao, et al., Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation, arXiv preprint arXiv:2311.07562 (2023).
- [90] H. Wen, Y. Li, G. Liu, S. Zhao, T. Yu, T. J.-J. Li, S. Jiang, Y. Liu, Y. Zhang, Y. Liu, Autodroid: Llm-powered task automation in android, in: Proceedings of the 30th Annual International Conference on Mobile Computing and Networking, 2024, pp. 543–557.
- [91] M. D. Vu, H. Wang, Z. Li, J. Chen, S. Zhao, Z. Xing, C. Chen, Gptvoicetasker: Llm-powered virtual assistant for smartphone, arXiv preprint arXiv:2401.14268 (2024).
- [92] M. Santacrose, Z. Wen, Y. Shen, Y. Li, What Matters In The Structured Pruning of Generative Language Models?, 2023. [arXiv:2302.03773](https://arxiv.org/abs/2302.03773).

- [93] X. Ma, G. Fang, X. Wang, LLM-Pruner: On the Structural Pruning of Large Language Models, 2023. [arXiv:2305.11627](#).
- [94] T. Chen, T. Ding, B. Yadav, I. Zharkov, L. Liang, LoRAShear: Efficient Large Language Model Structured Pruning and Knowledge Recovery, 2023. [arXiv:2310.18356](#).
- [95] M. Zhang, H. Chen, C. Shen, Z. Yang, L. Ou, X. Yu, B. Zhuang, LoRAPrune: Pruning Meets Low-Rank Parameter-Efficient Fine-Tuning, 2023. [arXiv:2305.18403](#).
- [96] H. Wang, Z. Zhang, S. Han, SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning, in: 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA), IEEE Computer Society, 2021, pp. 97–110.
- [97] E. Kurtic, E. Frantar, D. Alistarh, ZipLM: Inference-Aware Structured Pruning of Language Models, 2023. [arXiv:2302.04089](#).
- [98] E. Frantar, D. Alistarh, SparseGPT: Massive Language Models Can be Accurately Pruned in One-Shot, in: Proceedings of the 40th International Conference on Machine Learning, volume 202 of *Proceedings of Machine Learning Research*, PMLR, New York, 2023, pp. 10323–10337.
- [99] M. Sun, Z. Liu, A. Bair, J. Z. Kolter, A Simple and Effective Pruning Approach for Large Language Models, 2023. [arXiv:2306.11695](#).
- [100] H. Shao, B. Liu, B. Xiao, K. Zeng, G. Wan, Y. Qian, One-Shot Sensitivity-Aware Mixed Sparsity Pruning for Large Language Models, 2024. [arXiv:2310.09499](#).
- [101] H. Xia, Z. Zheng, Y. Li, D. Zhuang, Z. Zhou, X. Qiu, Y. Li, W. Lin, S. L. Song, Flash-LLM: Enabling Cost-Effective and Highly-Efficient Large Generative Model Inference with Unstructured Sparsity, 2023. [arXiv:2309.10285](#).
- [102] R. Agarwal, N. Vieillard, Y. Zhou, P. Stanczyk, S. Ramos, M. Geist, O. Bachem, On-Policy Distillation of Language Models: Learning from Self-Generated Mistakes, 2024. [arXiv:2306.13649](#).

- [103] C. Liang, S. Zuo, Q. Zhang, P. He, W. Chen, T. Zhao, Less is More: Task-aware Layer-wise Distillation for Language Model Compression, in: Proceedings of the 40th International Conference on Machine Learning, PMLR, New York, 2023, pp. 20852–20867.
- [104] X. Zhu, B. Qi, K. Zhang, X. Long, B. Zhou, PaD: Program-aided Distillation Specializes Large Models in Reasoning, 2023. [arXiv:2305.13888](https://arxiv.org/abs/2305.13888).
- [105] L. H. Li, J. Hessel, Y. Yu, X. Ren, K.-W. Chang, Y. Choi, Symbolic Chain-of-Thought Distillation: Small Models Can Also “Think” Step-by-Step, in: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, volume 1, Association for Computational Linguistics, Toronto, Canada, 2023, pp. 2665–2679.
- [106] N. Ho, L. Schmid, S.-Y. Yun, Large Language Models Are Reasoning Teachers, in: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, volume 1, Association for Computational Linguistics, Toronto, Canada, 2023, pp. 14852–14882.
- [107] P. Wang, Z. Wang, Z. Li, Y. Gao, B. Yin, X. Ren, SCOTT: Self-Consistent Chain-of-Thought Distillation, in: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, volume 1, Association for Computational Linguistics, Toronto, Canada, 2023, pp. 5546–5558.
- [108] C.-Y. Hsieh, C.-L. Li, C.-k. Yeh, H. Nakhost, Y. Fujii, A. Ratner, R. Krishna, C.-Y. Lee, T. Pfister, Distilling Step-by-Step! Outperforming Larger Language Models with Less Training Data and Smaller Model Sizes, in: Findings of the Association for Computational Linguistics: ACL 2023, Association for Computational Linguistics, Toronto, Canada, 2023, pp. 8003–8017.
- [109] Z. Chen, Q. Gao, A. Bosselut, A. Sabharwal, K. Richardson, DISCO: Distilling Counterfactuals with Large Language Models, in: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, volume 1, Association for Computational Linguistics, Toronto, Canada, 2023, pp. 5514–5528.
- [110] Y. Jiang, C. Chan, M. Chen, W. Wang, Lion: Adversarial distillation of proprietary large language models, in: Proceedings of the 2023 Conference

on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Singapore, 2023, pp. 3134–3154.

- [111] X. Xu, M. Li, C. Tao, T. Shen, R. Cheng, J. Li, C. Xu, D. Tao, T. Zhou, A Survey on Knowledge Distillation of Large Language Models, 2024. [arXiv:2402.13116](#).
- [112] Y. Huang, Y. Chen, Z. Yu, K. McKeown, In-context Learning Distillation: Transferring Few-shot Learning Ability of Pre-trained Language Models, 2022. [arXiv:2212.10670](#).
- [113] X. Zhu, J. Li, Y. Liu, C. Ma, W. Wang, A Survey on Model Compression for Large Language Models, 2023. [arXiv:2308.07633](#).
- [114] Z. Liu, B. Oguz, C. Zhao, E. Chang, P. Stock, Y. Mehdad, Y. Shi, R. Krishnamoorthi, V. Chandra, LLM-QAT: Data-Free Quantization Aware Training for Large Language Models, 2023. [arXiv:2305.17888](#).
- [115] M. Chen, W. Shao, P. Xu, J. Wang, P. Gao, K. Zhang, Y. Qiao, P. Luo, EfficientQAT: Efficient Quantization-Aware Training for Large Language Models, 2024. [arXiv:2407.11062](#).
- [116] X. Shen, Z. Kong, C. Yang, Z. Han, L. Lu, P. Dong, C. Lyu, C.-h. Li, X. Guo, Z. Shu, W. Niu, M. Leeser, P. Zhao, Y. Wang, EdgeQAT: Entropy and Distribution Guided Quantization-Aware Training for the Acceleration of Lightweight LLMs on the Edge, 2024. [arXiv:2402.10787](#).
- [117] J. Lin, J. Tang, H. Tang, S. Yang, X. Dang, C. Gan, S. Han, AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration, 2023. [arXiv:2306.00978](#).
- [118] E. Frantar, S. Ashkboos, T. Hoefler, D. Alistarh, OPTQ: Accurate quantization for generative pre-trained transformers, in: Proceedings of the 11th International Conference on Learning Representations, OpenReview.net, New York, 2023.
- [119] T. Dettmers, R. Svirschevski, V. Egiazarian, D. Kuznedelev, E. Frantar, S. Ashkboos, A. Borzunov, T. Hoefler, D. Alistarh, SpQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression, 2023. [arXiv:2306.03078](#).

- [120] C. Lee, J. Jin, T. Kim, H. Kim, E. Park, OWQ: Lessons learned from activation outliers for weight quantization in large language models, 2023. [arXiv:2306.02272](https://arxiv.org/abs/2306.02272).
- [121] V. Egiazarian, A. Panferov, D. Kuznedelev, E. Frantar, A. Babenko, D. Alistarh, Extreme Compression of Large Language Models via Additive Quantization, in: Forty-first International Conference on Machine Learning, 2024.
- [122] T. Dettmers, M. Lewis, Y. Belkada, L. Zettlemoyer, GPT3.int8(): 8-bit Matrix Multiplication for Transformers at Scale, *Advances in Neural Information Processing Systems* 35 35 (2022) 30318–30332.
- [123] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, S. Han, SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models, in: *Proceedings of the 40th International Conference on Machine Learning*, PMLR, 2023, pp. 38087–38099.
- [124] Z. Yao, R. Yazdani Aminabadi, M. Zhang, X. Wu, C. Li, Y. He, ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers, *Advances in Neural Information Processing Systems* 35 35 (2022) 27168–27183.
- [125] X. Wei, Y. Zhang, Y. Li, X. Zhang, R. Gong, J. Guo, X. Liu, Outlier Suppression+: Accurate quantization of large language models by equivalent and effective shifting and scaling, in: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, Singapore, 2023, pp. 1648–1665.
- [126] X. Wei, Y. Zhang, X. Zhang, R. Gong, S. Zhang, Q. Zhang, F. Yu, X. Liu, Outlier Suppression: Pushing the Limit of Low-bit Transformer Language Models, *Advances in Neural Information Processing Systems* 35 35 (2022) 17402–17414.
- [127] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, F. Yang, Y. Liu, M. Guo, Y. Zhu, OliVe: Accelerating Large Language Models via Hardware-friendly Outlier-Victim Pair Quantization, in: *Proceedings of the 50th Annual International Symposium on Computer Architecture, ISCA '23*, Association for Computing Machinery, New York, 2023, pp. 1–15.

- [128] J. Liu, R. Gong, X. Wei, Z. Dong, J. Cai, B. Zhuang, QLLM: Accurate and Efficient Low-Bitwidth Quantization for Large Language Models, 2023. [arXiv:2310.08041](#).
- [129] Q. Li, Y. Zhang, L. Li, P. Yao, B. Zhang, X. Chu, Y. Sun, L. Du, Y. Xie, FPTQ: Fine-grained Post-Training Quantization for Large Language Models, 2023. [arXiv:2308.15987](#).
- [130] X. Shen, P. Dong, L. Lu, Z. Kong, Z. Li, M. Lin, C. Wu, Y. Wang, Agile-Quant: Activation-Guided Quantization for Faster Inference of LLMs on the Edge, Proceedings of the AAAI Conference on Artificial Intelligence 38 (2024) 18944–18951.
- [131] Y. Shang, Z. Yuan, Q. Wu, Z. Dong, PB-LLM: Partially Binarized Large Language Models, 2023. [arXiv:2310.00034](#).
- [132] W. Huang, Y. Liu, H. Qin, Y. Li, S. Zhang, X. Liu, M. Magno, X. Qi, BiLLM: Pushing the Limit of Post-Training Quantization for LLMs, 2024. [arXiv:2402.04291](#).
- [133] H. Wang, S. Ma, L. Dong, S. Huang, H. Wang, L. Ma, F. Yang, R. Wang, Y. Wu, F. Wei, BitNet: Scaling 1-bit Transformers for Large Language Models, 2023. [arXiv:2310.11453](#).
- [134] S. Ma, H. Wang, L. Ma, L. Wang, W. Wang, S. Huang, L. Dong, R. Wang, J. Xue, F. Wei, The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits, 2024. [arXiv:2402.17764](#).
- [135] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, LoRA: Low-Rank Adaptation of Large Language Models, in: Proceedings of the 10th International Conference on Learning Representations, OpenReview.net, New York, 2022.
- [136] A. Kaushal, T. Vaidhya, I. Rish, LORD: Low Rank Decomposition Of Monolingual Code LLMs For One-Shot Compression, 2023. [arXiv:2309.14021](#).
- [137] Y. Li, Y. Yu, Q. Zhang, C. Liang, P. He, W. Chen, T. Zhao, LoSparse: Structured Compression of Large Language Models based on Low-Rank and Sparse Approximation, in: Proceedings of the 40th International Conference on Machine Learning, PMLR, New York, 2023, pp. 20336–20350.

- [138] M. Xu, Y. L. Xu, D. P. Mandic, TensorGPT: Efficient Compression of the Embedding Layer in LLMs based on the Tensor-Train Decomposition, 2023. [arXiv:2307.00526](#).
- [139] C. Chen, S. Borgeaud, G. Irving, J.-B. Lespiau, L. Sifre, J. Jumper, Accelerating Large Language Model Decoding with Speculative Sampling, 2023. [arXiv:2302.01318](#).
- [140] Y. Leviathan, M. Kalman, Y. Matias, Fast inference from transformers via speculative decoding, in: Proceedings of the 40th International Conference on Machine Learning, volume 202 of *ICML'23*, JMLR.org, Honolulu, Hawaii, USA, 2023, pp. 19274–19286.
- [141] S. Kim, K. Mangalam, S. Moon, J. Malik, M. W. Mahoney, A. Gholami, K. Keutzer, Speculative Decoding with Big Little Decoder, *Advances in Neural Information Processing Systems* 36 (2023) 39236–39256.
- [142] X. Miao, G. Oliaro, Z. Zhang, X. Cheng, Z. Wang, Z. Zhang, R. Y. Y. Wong, A. Zhu, L. Yang, X. Shi, C. Shi, Z. Chen, D. Arfeen, R. Abhyankar, Z. Jia, SpecInfer: Accelerating Large Language Model Serving with Tree-based Speculative Inference and Verification, in: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, volume 3 of *ASPLOS '24*, Association for Computing Machinery, New York, NY, USA, 2024, pp. 932–949.
- [143] Y. Li, F. Wei, C. Zhang, H. Zhang, EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty, 2024. [arXiv:2401.15077](#).
- [144] T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, T. Dao, Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads, 2024. [arXiv:2401.10774](#).
- [145] Y. Fu, P. Bailis, I. Stoica, H. Zhang, Break the Sequential Dependency of LLM Inference Using Lookahead Decoding, 2024. [arXiv:2402.02057](#).
- [146] D. Xu, W. Yin, X. Jin, Y. Zhang, S. Wei, M. Xu, X. Liu, LLMcad: Fast and Scalable On-device Large Language Model Inference, 2023. [arXiv:2309.04255](#).

- [147] R. Svirschevski, A. May, Z. Chen, B. Chen, Z. Jia, M. Ryabinin, SpecExec: Massively Parallel Speculative Decoding for Interactive LLM Inference on Consumer Devices, 2024. [arXiv:2406.02532](#).
- [148] C. Hooper, S. Kim, H. Mohammadzadeh, M. W. Mahoney, Y. S. Shao, K. Keutzer, A. Gholami, KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization, 2024. [arXiv:2401.18079](#).
- [149] S. Ge, Y. Zhang, L. Liu, M. Zhang, J. Han, J. Gao, Model Tells You What to Discard: Adaptive KV Cache Compression for LLMs, in: The Twelfth International Conference on Learning Representations, 2023.
- [150] G. Xiao, Y. Tian, B. Chen, S. Han, M. Lewis, Efficient Streaming Language Models with Attention Sinks, in: The Twelfth International Conference on Learning Representations, 2023.
- [151] Z. Liu, A. Desai, F. Liao, W. Wang, V. Xie, Z. Xu, A. Kyrillidis, A. Shrivastava, Scissorhands: exploiting the persistence of importance hypothesis for LLM KV cache compression at test time, in: Proceedings of the 37th International Conference on Neural Information Processing Systems, NIPS '23, Curran Associates Inc., Red Hook, NY, USA, 2024, pp. 52342–52364.
- [152] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett, Z. A. Wang, B. Chen, H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models, *Advances in Neural Information Processing Systems* 36 (2023) 34661–34710.
- [153] Z. Liu, J. Yuan, H. Jin, S. Zhong, Z. Xu, V. Braverman, B. Chen, X. Hu, KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache, 2023. [arXiv:2402.02750](#).
- [154] S. Dong, W. Cheng, J. Qin, W. Wang, QAQ: Quality Adaptive Quantization for LLM KV Cache, 2024. [arXiv:2403.04643](#).
- [155] R. Liu, H. Bai, H. Lin, Y. Li, H. Gao, Z. Xu, L. Hou, J. Yao, C. Yuan, IntactKV: Improving Large Language Model Quantization by Keeping Pivot Tokens Intact, 2024. [arXiv:2403.01241](#).
- [156] T. Schuster, A. Fisch, J. Gupta, M. Dehghani, D. Bahri, V. Tran, Y. Tay, D. Metzler, Confident Adaptive Language Modeling, *Advances in Neural Information Processing Systems* 35 (2022) 17456–17472.

- [157] Z. Zeng, Y. Hong, H. Dai, H. Zhuang, C. Chen, ConsistentEE: A Consistent and Hardness-Guided Early Exiting Method for Accelerating Language Models Inference, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 38, 2024, pp. 19506–19514.
- [158] L. Del Corro, A. Del Giorno, S. Agarwal, B. Yu, A. Awadallah, S. Mukherjee, SkipDecode: Autoregressive Skip Decoding with Batching and Caching for Efficient LLM Inference, 2023. [arXiv:2307.02628](https://arxiv.org/abs/2307.02628).
- [159] Y. Chen, X. Pan, Y. Li, B. Ding, J. Zhou, EE-LLM: Large-Scale Training and Inference of Early-Exit Large Language Models with 3D Parallelism, 2024. [arXiv:2312.04916](https://arxiv.org/abs/2312.04916).
- [160] S. Bae, J. Ko, H. Song, S.-Y. Yun, Fast and Robust Early-Exiting Framework for Autoregressive Language Models with Synchronized Parallel Decoding, in: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Singapore, 2023, pp. 5910–5924.
- [161] M. Elhoushi, A. Shrivastava, D. Liskovich, B. Hosmer, B. Wasti, L. Lai, A. Mahmoud, B. Acun, S. Agarwal, A. Roman, A. A. Aly, B. Chen, C.-J. Wu, LayerSkip: Enabling Early Exit Inference and Self-Speculative Decoding, 2024. [arXiv:2404.16710](https://arxiv.org/abs/2404.16710).
- [162] T. Dao, D. Fu, S. Ermon, A. Rudra, C. Ré, FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness, Advances in Neural Information Processing Systems 35 (2022) 16344–16359.
- [163] T. Dao, FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning, 2023. [arXiv:2307.08691](https://arxiv.org/abs/2307.08691).
- [164] T. Dao, D. Haziza, F. Massa, G. Sizov, Flash-Decoding for long-context inference, ??? URL: <https://pytorch.org/blog/flash-decoding/>, (Accessed: Feb. 3, 2024).
- [165] K. Hong, G. Dai, J. Xu, Q. Mao, X. Li, J. Liu, K. Chen, Y. Dong, Y. Wang, FlashDecoding++: Faster Large Language Model Inference on GPUs, 2024. [arXiv:2311.01282](https://arxiv.org/abs/2311.01282).
- [166] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, I. Stoica, Efficient Memory Management for Large Language

- Model Serving with PagedAttention, in: Proceedings of the 29th Symposium on Operating Systems Principles, SOSP '23, Association for Computing Machinery, New York, 2023, pp. 611–626.
- [167] J. Wei, S. Cao, T. Cao, L. Ma, L. Wang, Y. Zhang, M. Yang, T-MAC: CPU Renaissance via Table Lookup for Low-Bit LLM Deployment on Edge, 2024. [arXiv:2407.00088](https://arxiv.org/abs/2407.00088).
 - [168] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, et al., Transformers: State-of-the-Art Natural Language Processing, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Association for Computational Linguistics, Online, 2020, pp. 38–45.
 - [169] G. Gerganov, ggerganov/llama.cpp, ??? URL: <https://github.com/ggerganov/llama.cpp>, (Accessed: Jun. 26, 2024).
 - [170] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen, P. Liang, C. Re, I. Stoica, C. Zhang, FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU, in: Proceedings of the 40th International Conference on Machine Learning, PMLR, New York, 2023, pp. 31094–31116.
 - [171] S. Chen, Y. Lin, M. Zhang, Y. Wu, Efficient and Economic Large Language Model Inference with Attention Offloading, 2024. [arXiv:2405.01814](https://arxiv.org/abs/2405.01814).
 - [172] Z. Liu, J. Wang, T. Dao, T. Zhou, B. Yuan, Z. Song, A. Shrivastava, C. Zhang, Y. Tian, C. Re, B. Chen, Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time, in: Proceedings of the 40th International Conference on Machine Learning, PMLR, New York, 2023, pp. 22137–22176.
 - [173] Y. Song, Z. Mi, H. Xie, H. Chen, PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU, 2023. [arXiv:2312.12456](https://arxiv.org/abs/2312.12456).
 - [174] K. Alizadeh, I. Mirzadeh, D. Belenko, K. Khatamifard, M. Cho, C. C. Del Mundo, M. Rastegari, M. Farajtabar, LLM in a flash: Efficient Large Language Model Inference with Limited Memory, 2024. [arXiv:2312.11514](https://arxiv.org/abs/2312.11514).

- [175] R. Yi, L. Guo, S. Wei, A. Zhou, S. Wang, M. Xu, EdgeMoE: Fast On-Device Inference of MoE-based Large Language Models, 2023. [arXiv:2308.14352](https://arxiv.org/abs/2308.14352).
- [176] NVIDIA/TensorRT-LLM, ??? URL: <https://github.com/NVIDIA/TensorRT-LLM>, (Accessed: Jun. 29, 2024).
- [177] R. Y. Aminabadi, S. Rajbhandari, A. A. Awan, C. Li, D. Li, E. Zheng, O. Ruwase, S. Smith, M. Zhang, J. Rasley, Y. He, DeepSpeed-Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale, in: SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, 2022, pp. 1–15.
- [178] huggingface/text-generation-inference, ??? URL: <https://github.com/huggingface/text-generation-inference>, (Accessed: Jun. 29, 2024).
- [179] A. Pham, C. Yang, S. Sheng, S. Zhao, S. Lee, B. Jiang, F. Dong, X. Guan, F. Ming, OpenLLM: Operating LLMs in production, ??? URL: <https://github.com/bentoml/OpenLLM>, (Accessed: Jun. 29, 2024).
- [180] InternLM/lmdeploy, ??? URL: <https://github.com/InternLM/lmdeploy>, (Accessed: Jun. 29, 2024).
- [181] PyTorch ExecuTorch, ??? URL: <https://pytorch.org/executorch-overview>, (Accessed: Jun. 27, 2024).
- [182] TensorFlow Lite | ML for Mobile and Edge Devices, ??? URL: <https://www.tensorflow.org/lite>, (Accessed: May 21, 2024).
- [183] alibaba/MNN, ??? URL: <https://github.com/alibaba/MNN>, (Accessed: Jun. 29, 2024).
- [184] H. Ni, The ncnn contributors, ncnn, ??? URL: <https://github.com/Tencent/ncnn>, (Accessed: Jun. 29, 2024).
- [185] Z. Xue, Y. Song, Z. Mi, L. Chen, Y. Xia, H. Chen, PowerInfer-2: Fast Large Language Model Inference on a Smartphone, 2024. [arXiv:2406.06282](https://arxiv.org/abs/2406.06282).
- [186] L. Li, S. Qian, J. Lu, L. Yuan, R. Wang, Q. Xie, Transformer-Lite: High-efficiency Deployment of Large Language Models on Mobile Phone GPUs, 2024. [arXiv:2403.20041](https://arxiv.org/abs/2403.20041).

- [187] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, T. Abdelzaher, Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency, in: Proceedings of the 18th Conference on Embedded Networked Sensor Systems, SenSys '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 476–488.
- [188] Z. Zhao, K. Wang, N. Ling, G. Xing, EdgeML: An AutoML Framework for Real-Time Deep Learning on the Edge, in: Proceedings of the International Conference on Internet-of-Things Design and Implementation, IoTDI '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 133–144.
- [189] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, N. D. Lane, SPINN: Synergistic progressive inference of neural networks over device and cloud, in: Proceedings of the 26th Annual International Conference on Mobile Computing and Networking, MobiCom '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1–15.
- [190] K. Huang, W. Gao, Real-time neural network inference on extremely weak devices: Agile offloading with explainable AI, in: Proceedings of the 28th Annual International Conference on Mobile Computing And Networking, MobiCom '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 200–213.
- [191] D. Ding, A. Mallick, C. Wang, R. Sim, S. Mukherjee, V. Rühle, L. V. S. Lakshmanan, A. H. Awadallah, Hybrid LLM: Cost-Efficient and Quality-Aware Query Routing, in: The Twelfth International Conference on Learning Representations, 2023.
- [192] Y. Wang, K. Chen, H. Tan, K. Guo, Tabi: An Efficient Multi-Level Inference System for Large Language Models, in: Proceedings of the Eighteenth European Conference on Computer Systems, EuroSys '23, Association for Computing Machinery, New York, NY, USA, 2023, pp. 233–248.
- [193] F. Cai, D. Yuan, Z. Yang, L. Cui, Edge-llm: A collaborative framework for large language model serving in edge computing, in: 2024 IEEE International Conference on Web Services (ICWS), IEEE, 2024, pp. 799–809.
- [194] Z. Wu, S. Sun, Y. Wang, M. Liu, B. Gao, Q. Pan, T. He, X. Jiang, Agglomerative federated learning: Empowering larger model training via end-

- edge-cloud collaboration, in: IEEE INFOCOM 2024-IEEE Conference on Computer Communications, IEEE, 2024, pp. 131–140.
- [195] mlc-ai/mlc-llm, ??? URL: <https://github.com/mlc-ai/mlc-llm>, (Accessed: Feb. 4, 2024).
 - [196] UbiquitousLearning/mlm, ??? URL: <https://github.com/UbiquitousLearning/mlm>, (Accessed: Feb. 4, 2024).
 - [197] D. Franklin, dusty-nv/NanoLLM, ??? URL: <https://github.com/dusty-nv/NanoLLM>, (Accessed: Jun. 30, 2024).
 - [198] Z. Hao, H. Jiang, S. Jiang, J. Ren, T. Cao, Hybrid SLM and LLM for Edge-Cloud Collaborative Inference, in: Proceedings of the Workshop on Edge and Mobile Foundation Models, EdgeFM '24, Association for Computing Machinery, New York, NY, USA, 2024, pp. 36–41.
 - [199] G. Gerganov, ggerganov/ggml, ??? URL: <https://github.com/ggerganov/ggml>, (Accessed: Jun. 30, 2024).
 - [200] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, Z. Chen, GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding, in: International Conference on Learning Representations, 2020.
 - [201] B. Peng, E. Alcaide, Q. Anthony, A. Albalak, S. Arcadinho, S. Biderman, H. Cao, X. Cheng, M. Chung, L. Derczynski, X. Du, M. Grella, K. Gv, X. He, H. Hou, P. Kazienko, J. Kocon, J. Kong, B. Koptyra, H. Lau, J. Lin, K. S. I. Mantri, F. Mom, A. Saito, G. Song, X. Tang, J. Wind, S. Woźniak, Z. Zhang, Q. Zhou, J. Zhu, R.-J. Zhu, RWKV: Reinventing RNNs for the Transformer Era, in: Findings of the Association for Computational Linguistics: EMNLP 2023, Association for Computational Linguistics, Singapore, 2023, pp. 14048–14077.
 - [202] A. Gu, T. Dao, Mamba: Linear-Time Sequence Modeling with Selective State Spaces, 2024. [arXiv:2312.00752](https://arxiv.org/abs/2312.00752).
 - [203] Y. Sun, L. Dong, S. Huang, S. Ma, Y. Xia, J. Xue, J. Wang, F. Wei, Re-tentive Network: A Successor to Transformer for Large Language Models, 2023. [arXiv:2307.08621](https://arxiv.org/abs/2307.08621).

- [204] W. Yin, M. Xu, Y. Li, X. Liu, LLM as a System Service on Mobile Devices, 2024. [arXiv:2403.11805](https://arxiv.org/abs/2403.11805).
- [205] G. Heo, S. Lee, J. Cho, H. Choi, S. Lee, H. Ham, G. Kim, D. Mahajan, J. Park, NeuPIMs: NPU-PIM Heterogeneous Acceleration for Batched LLM Inferencing, in: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, volume 3 of *ASPLOS '24*, Association for Computing Machinery, New York, NY, USA, 2024, pp. 722–737.
- [206] Y. Hu, C. Imes, X. Zhao, S. Kundu, P. A. Beerel, S. P. Crago, J. P. Walters, PipeEdge: Pipeline Parallelism for Large-Scale Model Inference on Heterogeneous Edge Devices, in: 2022 25th Euromicro Conference on Digital System Design (DSD), 2022, pp. 298–307.
- [207] Y. Jiang, R. Yan, X. Yao, Y. Zhou, B. Chen, B. Yuan, HexGen: Generative Inference of Large Language Model over Heterogeneous Environment, in: Proceedings of the 41st International Conference on Machine Learning, PMLR, 2024, pp. 21946–21961.
- [208] Z. Xuanlei, B. Jia, H. Zhou, Z. Liu, S. Cheng, Y. You, HeteGen: Efficient Heterogeneous Parallel Inference for Large Language Models on Resource-Constrained Devices, Proceedings of Machine Learning and Systems 6 (2024) 162–172.
- [209] H. Zhang, J. Da, D. Lee, V. Robinson, C. Wu, W. Song, T. Zhao, P. Raja, D. Slack, Q. Lyu, S. Hendryx, R. Kaplan, M. Lunati, S. Yue, A Careful Examination of Large Language Model Performance on Grade School Arithmetic, 2024. [arXiv:2405.00332](https://arxiv.org/abs/2405.00332).
- [210] A. Salemi, S. Mysore, M. Bendersky, H. Zamani, Lamp: When large language models meet personalization, arXiv preprint arXiv:2304.11406 (2023).
- [211] S. Doddapaneni, K. Sayana, A. Jash, S. Sodhi, D. Kuzmin, User embedding model for personalized language prompting, arXiv preprint arXiv:2401.04858 (2024).
- [212] L. Guo, W. Choe, F. X. Lin, Sti: Turbocharge nlp inference at the edge via elastic pipelining, in: Proceedings of the 28th ACM International Confer-

ence on Architectural Support for Programming Languages and Operating Systems, Volume 2, 2023, pp. 791–803.

- [213] J. Yuan, C. Yang, D. Cai, S. Wang, X. Yuan, Z. Zhang, X. Li, D. Zhang, H. Mei, X. Jia, et al., Mobile foundation model as firmware, in: Proceedings of the 30th Annual International Conference on Mobile Computing and Networking, 2024, pp. 279–295.